Fast Similarity Matrix Profile for Music Analysis and Exploration

Diego F. Silva, Chin-Chia M. Yeh, Yan Zhu, Gustavo E. A. P. A. Batista, Eamonn Keogh

***Abstract*—Most algorithms for music data mining and retrieval analyze the similarity between feature sets extracted from the raw audio. A conventional approach to assess similarities within or between recordings is to create similarity matrices. However, this method requires quadratic space for each comparison and typically requires a costly post-processing of the matrix. We have recently proposed SiMPle, a powerful representation based on subsequence similarity join, which is applicable in several music analysis tasks. In this paper, we propose SiMPle-Fast a highly efficient method for exact computation of SiMPle that is up to one order of magnitude faster than SiMPle. Furthermore, we demonstrate the utility of SiMPle-Fast in cover music recognition and thumbnailing tasks and show our method is significantly faster and more accurate than the state-of-the-art.**

***Index Terms*—Data analysis, distance measurement, music information retrieval**

I. Introduction

There exist an ever-increasing interest in applications related to music processing, music information retrieval and data mining in both academia and the music industry. However, the analysis of audio recordings remains a significant challenge, aggravated by the growing volume of music data created by the expansion of electronic music file distribution and streaming services. For this reason, algorithms for music analysis must be efficient in both time and space.

Most algorithms for content-based music analysis have at their cores some similarity or distance function. Consequently, a variety of applications rely on techniques to assess the similarity between music objects. These applications include: segmentation [1], audio-to-score alignment [2], cover song recognition [3], query by singing/humming [4], artist similarity identification [5], and visualization [6].

A standard approach to assessing similarity in music recordings is utilizing a self-similarity matrix (SSM) [7]. This representation reveals the relationship between each segment of a track to all the other segments in the same recording. This idea has been generalized to measure the relationships between subsequences of different songs, such as the application of cross-recurrence analysis for cover song recognition [8].

Because similarity matrices simultaneously reveal both the global and local structure of music recordings, they are highly advantageous. However, this representation requires quadratic space concerning the length of the feature vector used to describe the audio. For this reason, most methods used to find patterns in the similarity matrix are, at least, quadratic in time complexity. However, most information contained in similarity matrices is irrelevant or has little impact on their analysis. This observation suggests a room for improvements in representing music recordings regarding space and time efficiency.

In our previous work [9], we applied the all-pairs-similarity-search of subsequences, which is also known as similarity join, to assess the similarity between audio recordings for Music Information Retrieval (MIR) tasks. Analogously to the similarity matrices, representing the entire subsequence join requires a quadratic space, and it also has a high time complexity, which is dependent upon the length of the subsequences to be joined. However, we demonstrate how to exploit a new data structure called a matrix profile, which allows for a space-efficient representation of the similarity join matrix between subsequences. Moreover, we can leverage recent optimizations in the all-neighbor search to compute the matrix profile efficiently [10]. For clarity, we refer to this representation as Similarity Matrix ProfiLE (SiMPle).

Figure 1 illustrates an example of two matrices representing the dissimilarities/distances within and between recordings and their relative SiMPle, which corresponds to the minimum value of each column in the matrices.

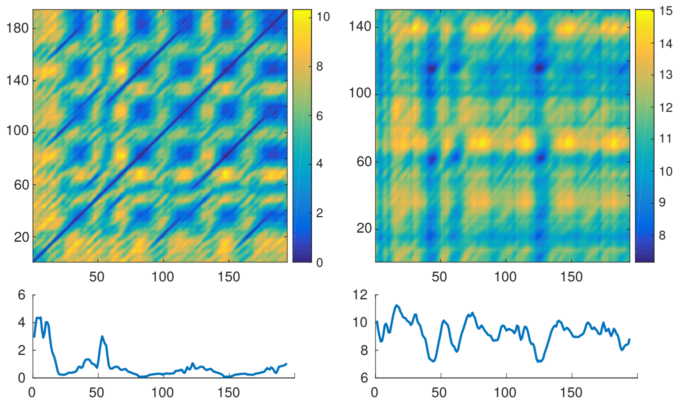


Fig. 1. Self-distance matrix (*left*), the cross-distance matrix between different recordings (*right*) and their respective SiMPle (*bottom*).

In this work, we extend our previous conference paper by further improving SiMPle time efficiency. We propose SiMPle-Fast a highly efficient method for exact computation of SiMPle that is up to one order of magnitude faster than SiMPle. We demonstrate the utility of SiMPle-Fast in cover music

Diego F. Silva is affiliated with both the Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Brazil and the Departamento de Computação, Universidade Federal de São Carlos, Brazil.

Gustavo E. A. P. A. Batista is affiliated with the Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Brazil.

Chin-Chia M. Yeh, Yan Zhu and Eamonn Keogh are affiliated with the Computer Science and Engineering at the University of California, Riverside, CA, USA.

Manuscript received April 19, 2005; revised August 26, 2015.

recognition and thumbnailing tasks and show our method is significantly faster and more accurate than the state-of-the-art.

The remainder of the paper is organized as follows. Section II introduces SiMPle and SiMPle-Fast, our novel algorithm to speed up the SiMPle calculation. Section III presents an experimental evaluation of SiMPle-Fast method, including the description of the used datasets (Section III-A), experiments on scalability (Section III-B), a broad experimentation on cover song recognition (Section III-C), and an algorithm based on SiMPle-Fast for audio thumbnailing (Section III-E). Finally, Section IV briefly summarizes the information contained in the paper and concludes the work.

II. SiMPle and SiMPle-Fast: Similarity Matrix Profile

We begin this section by presenting the necessary definitions to introduce our method. We use the terms time series and subsequences to refer to the feature vectors that describe the whole audio and excerpts respectively. Formally, we define a time series as follow.

Definition 1. A time series $T = (t_1, t_2, \ldots, t_n)$ is a contiguous sequence of vectors t_i with length n, such that t_i is composed of f real values comprising the features extracted to represent a small segment of the audio.

For clarity, if we are using chromatic features[1] to describe our audio files, f is the number of bins adopted (usually 12, 24, or 32) and n is the number of windows used to extract the features. Next, we define a subsequence.

Definition 2. A subsequence $S_s = (t_s, t_{s+1}, , t_{s+m-1})$ is a contiguous subset from the time series T with length m.

The primary operation for producing the similarity matrix profile is the similarity join, which is defined below.

Definition 3. Similarity join or AB-similarity join. Given two time series A and B and the desired subsequence length m, the similarity join identifies the nearest neighbor of each subsequence in A from all possible subsequences of B (both with length m).

Through a similarity join, we can gather two pieces of information about each subsequence in A, which are *i)* the distance to its nearest neighbor in B and *ii)* the position of its nearest neighbor in B. This information can be compactly stored in vectors, referred to as a similarity matrix profile (SiMPle) and similarity matrix profile index (SiMPle index), respectively.

Definition 4. Similarity matrix profile (SiMPle). Given two time series A and B, SiMPle is a vector representation of the AB-similarity join. The position P_i stores the distance between the subsequence A_i and its nearest neighbor in B. SiMPle index stores in I_i the index j of the subsequence in B_j which is the nearest neighbor of A_i.

[1]Despite that we used chromatic features in this work, our method is applicable to any set of features.

When both input time series refer to the same recording, this is a particular case of similarity join. We define the operation that handles this specific case as a self-similarity join.

Definition 5. Self-similarity join. Given a time series A and the desired subsequence length m, the self-similarity join identifies the nearest neighbor of each subsequence from A to every (non-trivial) subsequence set from A.

The only major difference between the self-similarity join (Definition 5) and the AB-similarity join (Definition 3) is the exclusion of trivially matched pairs when identifying the nearest neighbor. The exclusion of trivial matches is crucial, since matching a subsequence with itself (or slightly shifted version of itself) produces no useful information.

Definition 6. Trivial match. Given a subsequence A_i and its nearest neighbor A_j from a same time series A, a trivial match occurs when $i = j$. We also consider trivial matches the cases where $|i - j| < \xi$, for a parameter $\xi \leq m$. Trivial match subsequences share a significant amount of observations and naturally have small distances between them.

Given the relevant definitions, we are in the position to describe the original method to calculate SiMPle [9] in the Algorithm 1. We will later use this algorithm to highlight the differences to our proposal, SiMPle-Fast.

In line 1, we record the length of A. In line 2, we allocate memory and initialize SiMPle P and SiMPle index I. From line 3 to line 6, we calculate the distance profile vector D, which contains the distances between the subsequence in the time series A starting at idx and each subsequence in the time series B. In other words, for each subsequence of length m in A, we calculate a vector D which stores in the position i the distance between the assessed subsequence and the subsequence $B[i : i + m - 1], i \in \{1, 2, \ldots, length(B) - m + 1\}$.

The particular function we use to compute D is MASS, which is the most efficient algorithm known for distance vector computation [10]. Then we find the minimum value in D to update $P[idx]$, i.e., P in the position idx. We also update $I[idx]$ with the position of the minimum value in the vector D. Finally, we return P and I in line 7.

Algorithm 1 Procedure to calculate SiMPle and SiMPle index

```
Require: Two time series, $A$ and $B$, and the desired subsequence length $m$
Ensure: The SiMPle $P$ and the associated SiMPle index $I$
1: $len \leftarrow length(A) - m + 1$
2: $P \leftarrow infs(len), I \leftarrow zeros(len)$
3: for each $idx$ in $[1 : len]$ do
4:     $D \leftarrow MASS(A[idx : idx + m - 1], B)$ // c.f. [10]
5:     $P[idx], I[idx] \leftarrow min(D)$
6: end for
7: return $P, I$
```

Note that the Algorithm 1 computes SiMPle for the general similarity join. To modify it to compute the self-similarity join SiMPle of a time series A, we simply replace B by A in line 4 and ignore trivial matches in D when performing the min operation in line 5. To ignore trivial matches we ensure that $D[idx - \xi : idx + \xi - 1] = \infty$. This modification guarantees that the algorithm will not consider a subsequence starting in this interval as the nearest neighbor of the currently assessed subsequence. In our experiments, we defined $\xi = m/4$.

The method MASS (used in line 4) is important to speed up the similarity calculations. The main contribution of MASS is the use of an efficient divide-and-conquer Fast Fourier Transform-based method to calculate the cross-correlation between a subsequence and a time series. The cross-correlation provides the sliding dot product between the elements of both sequences. In other words, it provides the dot products between the subsequence under analysis and each subsequence of the same length from the other time series, or the same time series, in the case of self-join. Figure 2 illustrates this idea.

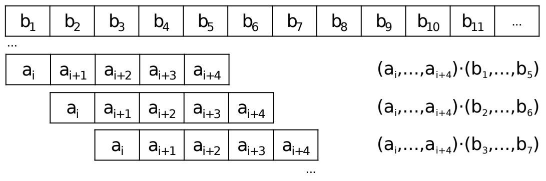

b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 ...
ai ai+1 ai+2 ai+3 ai+4
(ai,...,ai+4)·(b1,...,b5)
(ai,...,ai+4)·(b2,...,b6)
(ai,...,ai+4)·(b3,...,b7)

Fig. 2. The cross-correlation between a time series B and a subsequence from A provides the dot product between their elements.

This trick applies to several distance measures which depend on the dot product between values. In this work, we use the squared Euclidean distance. To clarify the use of the cross-correlation, Equation 1 defines the Euclidean distance, where S_A and S_B are subsequences from A and B, respectively.

$$\begin{aligned} ED(S_A, S_B) &= \sum_{i=1}^{m}(a_i - b_i)^2 \\ &= \sum_{i=1}^{m}(a_i)^2 + \sum_{i=1}^{m}(b_i)^2 - 2(S_A \cdot S_B) \end{aligned} \tag{1}$$

in which $S_A \cdot S_B$ is the dot product between S_A and S_B.

For each subsequence in A, we create a distance profile, which represents the distance between the currently assessed subsequence to every subsequence of the same length in B. Once we calculated the sliding dot product for this subsequence, Equation 1 provides the Euclidean distance to each position of the matrix profile.

The original algorithm to calculate SiMPle uses MASS for each subsequence. Recently, researchers noticed that the dot products do not need to be recalculated from scratch for each subsequence [11]. Instead, we can reuse the values calculated for the first subsequence to make a faster calculation in the next iterations. The idea is to make use of the intersections between the required products in consecutive iterations. Consider the subsequence $(a_i, a_{i+1}, \ldots, a_{i+m-1})$ of length m extracted from the time series A, starting at the position i. After the calculation of the distance profile to this subsequence, we have all the dot products $(a_i, a_{i+1}, \ldots, a_{i+m-1}) \cdot (b_j, b_{j+1}, \ldots, b_{j+m-1})$ for each valid j. By subtracting the first partial of these products, i.e., the value a_i multiplied by each value in the time series B, we obtain the products given by $(a_{i+1}, a_{i+2}, \ldots, a_{i+m-1}) \cdot (b_{j+1}, b_{j+2}, \ldots, b_{j+m-1})$. Finally, we can obtain $(a_{i+1}, a_{i+2}, \ldots, a_{i+m}) \cdot (b_{j+1}, b_{j+2}, \ldots, b_{j+m})$ by adding $(a_{i+m})(b_{j+m})$. The last dot product gives the required value to the next iteration. Figure 3 illustrates the procedure.

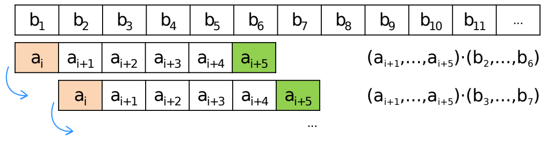

b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 ...
ai ai+1 ai+2 ai+3 ai+4 ai+5
(ai+1,...,ai+5)·(b2,...,b6)
(ai+1,...,ai+5)·(b3,...,b7)

Fig. 3. In order to quickly obtain the sliding dot product between a subsequence from the time series A and the time series B, our method reuses the products obtained from the previous subsequence in A.

After this procedure, we have all the values necessary to construct the distance profile that refers to the subsequence $(a_{i+1}, a_{i+2}, \ldots, a_{i+m})$, except for the first value. For this reason, before calculating the SiMPle, we calculate and store the cross-correlation of the first subsequence from B, i.e., $(b_1, b_2, \ldots, b_m)$ with the time series A. With this procedure, we obtain every necessary dot product to fulfill the distance profile calculations. Algorithm 2 presents this procedure in detail, which we refer as SiMPle-Fast.

Algorithm 2 SiMPle-Fast

```
Require: Two time series, $A$ and $B$, and the desired subsequence length $m$
Ensure: The SiMPle $P$ and the associated SiMPle index $I$
 1: $sqsumB, prodB \leftarrow initB(B, A)$
 2: $len \leftarrow length(A) - m + 1$
 3: $P \leftarrow infs(len), I \leftarrow zeros(len)$
 4: $D, prodA, sqsumA \leftarrow MASS(A[1:m], B)$
 5: $P[1], I[1] \leftarrow min(D)$
 6: $lp \leftarrow length(prodA)$
 7: for each $idx$ in $2 : len$ do
 8:     $sumval \leftarrow A[idx+m-1] \cdot B[len+1 : len+lp-1]$
 9:     $subval \leftarrow A[idx-1] \cdot B[1 : lp]$
10:     $prodA[2:lp] \leftarrow prodA[1:lp-1] + addval - subval$
11:     $prod[1] \leftarrow prodB[idx]$
12:     $sqsumA \leftarrow sqsumA - A[idx-1]^2 + A[idx+m-1]^2$
13:     $D \leftarrow sqsumA + sqsumB - 2 * prod$
14:     $P[idx], I[idx] \leftarrow min(D)$
15: end for
16: return $P, I$
```

To explain this algorithm, we will split into two main parts. The first one comprises lines 1 to 6. In this section, the algorithm calculates and stores auxiliary values and computes the first value of the SiMPle. Specifically, line 1 calculates the cumulative squared sum of all elements from B, as well as the sliding dot product between the first subsequence of B and the time series A. These values will be used to calculate the distances and update the sliding dot products regarding subsequences from A, respectively. The variables len and lp store the lengths of the SiMPle and the vector containing the products between a subsequence from A and the time series B. The algorithm uses these values in further operations. Finally, the operations between lines 3 and 5 compute the distance between the first subsequence from A to the time series B and store its minimum in the SiMPle. Note that we modified MASS to return the vector containing the dot products and the squared sum of elements in the subsequence from A.

In the second section, comprised between lines 7 and 15, the algorithm computes the remaining positions of SiMPle by updating the previously obtained values. It starts by updating the dot products, using the operations from line 8 to line 11. These operations perform the previously explained steps to reuse the dot products obtained from $(a_i, a_{i+1}, \ldots, a_{i+m-1})$ to obtain the products for $(a_{i+1}, a_{i+2}, \ldots, a_{i+m})$, as illustrated in Figure 3. Line 11 attributes the product between the first subsequence from B and the subsequence starting at the position idx in the time series A to the first position of the new vector of sliding dot product. Recall that the operation performed in line 1 stored this value.

Finally, line 12 updates the cumulative squared sum for the new subsequence, to use it in the distance calculation in line 13. Note that line 13 is the implementation of Equation 1. Once the distance is computed, we attribute its minimum value to its relative position in SiMPle.

III. Experimental Evaluation

This work proposes SiMPle-Fast, a fast and space-efficient method to assess local similarities within a music recording or between two different tracks. Our proposal applies to a multitude of music information retrieval and data mining tasks. For some examples, we refer the reader to [9].

In this paper, we focus on two straightforward algorithms to deal with the tasks of cover music recognition and audio thumbnail identification. We note that, although both algorithms use the SiMPle and SiMPle index computed by SiMPle-Fast, they have some differences between them.

In cover music recognition, we apply the AB-join and use the median of SiMPle distances to estimate the global distance between different recordings. Conversely, in thumbnail identification, we utilize the SiMPle index over the self-join operation to compute the most frequent nearest neighbor subsequence. We also use SiMPle to resolve draw cases, in which the latter operation equally scores two or more subsequences.

Before describing these methods and presenting results, we introduce the datasets used in our experimental analysis and experiments on the scalability of SiMPle-Fast.

A. Datasets

We evaluate our method in different scenarios regarding music styles and the size of the databases. Specifically, we assess SiMPle-Fast on popular and classical recordings. The first database considered is the YouTube Covers [12], which consists of 50 different compositions, each containing seven different recordings obtained from YouTube videos. The original data have training and testing partitions, in which the training set has one original studio recording and one live version performed by the same artist. For the cover song recognition task, we follow the same configuration of [12] to allow comparisons with literature results.

The second dataset we consider is the widely used collection of Chopin's Mazurkas [13]. The set of Mazurkas used in this work contains 2,919 recordings of 49 pieces for piano. The number of recordings of each song varies from 41 to 95.

The evaluation of different choices of feature sets is not the focus of this paper. For this reason, we fixed the use of chroma-based features in our experiments. This kind of features is prevalent in many music retrieval and data mining tasks. Specifically, we used the chroma energy normalized statistics (CENS) [14] to provide local tempo invariance. These features rely on applying a Hanning window on consecutive windows of "raw" chroma and aggregate features among the window to smooth the temporal chroma variation and reduce the dimensionality of the examples.

B. Scalability

Before we introduce the two approaches for cover song recognition and thumbnailing using the SiMPle and the SiMPle-index representations, we present an experiment to evaluate how the SiMPle-Fast improves the runtime for calculating these primitives.

The length of the time series and the number of tracks in the database directly affect the runtime of our method, as well as the runtime of any other method. For this reason, we performed one separate experiment for each of these parameters. In this section, we compare the proposed method against the original SiMPle calculation and the Dynamic Time Warping (DTW) [15]. Although we compare our method against other algorithms considered state-of-the-art for specific tasks in the next sections, these methods are at least as slow as DTW. For instance, both DTW and the local alignment distance proposed by Serrà et al. [3] have $\mathcal{O}(n^2)$ time complexity. However, DTW has smaller constant factors than Serrà's approach.

In our first experiment, we randomly chose one recording on the YouTube Covers dataset as a query and measured the similarity to all other tracks in the dataset. In the case of SiMPle, the similarity is the median distance between all subsequences of length m of the query and the time series under comparison. This approach is the same we use to calculate similarity for cover music recognition in Section III-C. After each distance calculation, we register the cumulative runtime of the experiment. This experiment shows how the runtime increases according to the size of the dataset, i.e., the number of tracks compared to the query. Figure 4 presents the results for two feature rates, 0.5 and 10 CENS per second, which results in different time series lengths.

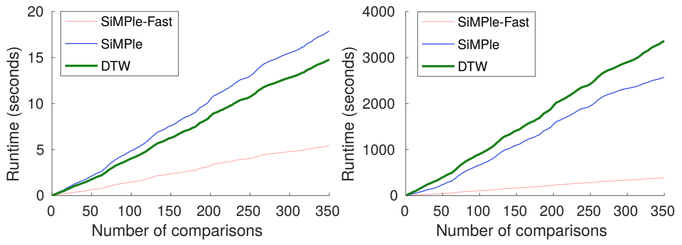


Fig. 4. Runtime obtained by querying one song by varying the number of objects in the dataset and the length of the time series. Specifically, we used 0.5 (*left*) and 10 (*right*) CENS per second in the feature extraction process.

The results show a linear behavior of the three methods with respect to the size of the data set. In any of these cases, the proposed method is faster than the others. Specifically, for the case of 10 CENS per second (around 1,600 features per song in average), the SiMPle-Fast is 8.5 times faster than DTW and 6.6 times faster than the original SiMPle implementation. In the case of using 0.5 CENS per second, resulting in an average of 132 CENS per song, our method is approximately 2.7 times faster than DTW and 3.3 times faster than SiMPle.

These are the smallest and highest numbers of features assessed in our experiments. An appropriate value for this parameter is task- and data-dependent. For the cover song recognition task, we found that 2 CENS per second was the most suitable value (c.f. Section III-C). This value leads to an average of 526 features per recording in the YouTube Covers data. In this case, the proposed method is around 5.1 times faster than both DTW and the original SiMPle algorithm. However, we notice that a larger number of features may be required in many tasks, mainly when we look for time precision, such as visualization and segmentation.

Another interesting aspect to observe is the relation between the runtime of SiMPle and DTW. Notice in Figure 4 that DTW outperforms SiMPle with 0.5 CENS per second, but is outperformed with 10 CENS per second. Moreover, both are outperformed by SiMPle-Fast.

Therefore, the method that performs best between DTW and SiMPle is dependent on the length of the time series. For this reason, we performed a second experiment, in which we evaluate the influence of the time series length on the runtime. For this, we use different feature rates, providing us the runtime for seven different lengths of time series. We executed each experiment five times and computed the average of the runtime values for each time series length. Figure 5 presents the results.

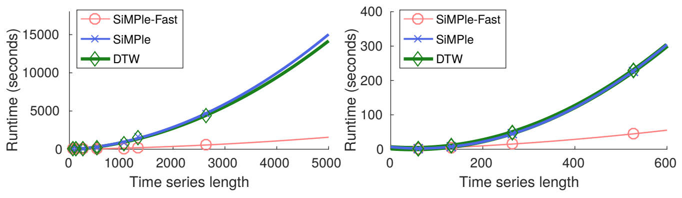


Fig. 5. Average runtime for comparing random queries to the remaining examples in the YouTube Covers dataset by varying the length of the time series (*left*), given by the number of features per second, and a zoom in the region of the shorter time series for the sake of better visualization of the runtime (*right*). The markers represent actual runtime measurements, and the lines represent values from a quadratic curve fitting interpolation.

In the second experiment, there is a quadratic trend on the runtime concerning the increasing length of the time series. Despite the fact that the original SiMPle and the DTW have a similar performance, in this case, the SiMPle-Fast is notably faster than both. More importantly, these results show that the longer the time series, the more significant the improvement provided by our method.

C. SiMPle-based Cover Song Recognition

"Cover song" is the generic term used to denote a new performance of a previously recorded track. For example, a cover song may refer to a live performance, a remix, or an interpretation in a different music style. The automatic identification of covers has several applications, such as copyright management, collection organization, and search by content.

In order to identify different versions of the same song, most algorithms search for globally [16] or locally [3], [17] conserved structure(s). A well-known and widely applied algorithm for measuring the global similarity between tracks is DTW. Despite its utility in other domains, frequently DTW is not flexible enough to handle the differences in structure between the recordings. A potential solution would be segmenting the song before applying the DTW similarity estimation. However, audio segmentation itself is also an open problem, and the error on boundary detection can cause a domino effect (compounded errors) in the identification process.

Also, the complexity of the algorithm to calculate DTW is $\mathcal{O}(n^2)$. Although there exist methods to approximate the DTW with fast algorithms [18], such methods lack an error bound for such approximations. In other words, these methods do not guarantee a maximum approximation error compared to the actual DTW.

Algorithms that search for local similarities have been successfully used to provide structural invariance to the cover song identification task. A well-known method for music similarity proposes the use of a binary distance function to compare chroma-based features, followed by a dynamic programming local alignment [3]. Despite its demonstrated utility to recognize cover recordings and some variants proposed in the literature, this method has several parameters that are unintuitive to tune and is computationally demanding. Specifically, the local alignment is estimated by an algorithm with similar complexity to DTW. Plus, the binary distance between chroma features used in each step of the algorithm relies on multiple shifts of the chroma vectors under comparison.

In this work, we propose to use SiMPle-Fast to measure the distance between recordings to identify cover songs. In essence, we exploit the fact that the global relation between the tracks is composed of many local similarities. In this way, we simultaneously take advantage of both local and global pattern matching.

Intuitively, we expect that the SiMPle obtained by comparing a cover song to its original version is composed mostly of low values. In contrast, two completely different songs will result in a SiMPle constituted mainly by high values. For this reason, we adopted the median value of the SiMPle as a global distance estimation. Formally, the distance between a query B and a candidate original recording A is defined in Equation 2.

$$d(A,B) = median(SiMPle(B,A)); \qquad (2)$$

It is important to notice that a cover song recognition algorithm needs to be robust for several distortions between different versions. While our method is robust to structural variations (c.f. Section III-C1) and we use CENS to improve the robustness against tempo variations, our distance measure is very sensitive to key differences. For this reason, we apply the Optimal Transposition Index (OTI) [19] to transpose the recordings to the same key.

Note that several other statistical measures could be used instead of the median. However, the median is robust enough to handle outliers in the matrix profile. Distortions may appear when a performer decides, for instance, to add a new segment (e.g., an improvisation or drum solo) to the song. The robustness of our method in this situation, as well as other changes in structure, is discussed in the next section.

1) On the Structural Invariance: The structural variance is a critical concern when comparing different songs. Changes in structure may occur by insertion or deletion of segments, as well as changes in the order that different excerpts are played. From a high-level point of view, SiMPle describes a global similarity outline between songs by providing information about local comparisons. This fact has several implications in our distance estimation, which makes it largely invariant to structural variations:

- If two performances are virtually identical, except for the order and the number of repetitions of each representative excerpt (i.e., chorus, verse, bridge, etc.), all the values that compose SiMPle are close to zero.
- If a segment of the original version is deleted in the cover song, this will cause virtually no changes in the SiMPle.
- If a new segment is inserted into a cover, the only consequence is a peak in the SiMPle, which may only slightly increase its median value.

We have experimented to testify these assumptions. For this, we compared the SiMPle-based distances between four songs in the YouTube Covers dataset and some variations of them. Specifically, the songs used in this experiment are the following:

- **Song A**: The first original recording (in alphabetical order) in the dataset, i.e., ABBA - "Dancing Queen."
- **Song B**: A cover version of the song A from the movie "Mamma Mia."
- **Song C**: The non-cover nearest neighbor of song A, which is a cover version of "No Woman No Cry," by Bob Marley.
- **Song D**: A randomly chosen song, which is which is a cover version of "I Want It That Way," by Backstreet Boys.

Other variations were proposed in this experiment to verify the assumptions earlier described in this section. They are:

- **Song A2**: The song A repeated twice, i.e., the concatenation of song A with itself.
- **Song A3**: The song A cut in half, i.e., truncated in the position which defines half of its length.
- **Song A4**: The song A with ten subsequences randomly displaced in time.
- **Song B2**: The song B with one randomly chosen subsequence of 20 seconds removed. The same strategy was used to create **C2** and **D2**.
- **Song B3**: The song B after inputting a random subsequence from D. The same strategy was used to create **C3**.

First, we evaluated the effects of modifying the original song A. We use A, A2, A3 and A4 as reference and compute the SiMPle-based distance to songs B, C, and D. Table I summarize the results.

TABLE I
DISTANCES FROM THREE DISTINCT QUERY TO FOUR VARIATIONS OF THE SAME REFERENCE RECORDINGS

Song identifier	A	A2	A3	A4
B	3.20	3.20	3.56	3.25
C	5.57	5.57	6.32	5.57
D	8.22	8.22	8.70	8.13

We also experimented with the variations of A as faithful versions of the original song, but with structural invariance. In this case, we computed the SiMPle-based distance from A to A2, A3, and A4. The distances to A2 and A4 are in the order of 10^{-14}, which may happen exclusively because of rounding errors. The distance between A and A3 is 0.615, which is significantly smaller than the distance calculated to the cover B, which is song A's nearest neighbor.

Finally, we examined the effects of modifying the recordings of cover versions. Table II shows the results. Note that in all these cases the distances do not change significantly.

TABLE II
DISTANCES FROM THE VARIATIONS OF THE QUERIES B, C, AND D TO THE ORIGINAL VERSION OF SONG A

Song identifier	B2	B3	C2	C3	D2
A	3.39	3.42	5.50	5.68	8.68

We also made a more extensive evaluation of SiMPle robustness to structural variations. We calculated the distance between each variation of song A, i.e., A, A2, A3, and A4, and all the remaining 349 recordings in the dataset. Then, we measured the Pearson correlation coefficient between the vectors of distances obtained using A and each of its variations as the original version. The correlations regarding A2, A3, and A4 are, respectively, 0.999, 0.980, and 0.989. This clearly shows that distinct structural differences barely affects our method.

2) Results and Discussion: We used three commonly applied evaluation measures to assess the performance of our method in the cover song recognition task: mean average precision (MAP), precision at 10 (P@10), and the mean rank of first correctly identified cover (MR1). Note that for MR1, smaller values are better.

Table III shows the results for the YouTube Covers. In addition to comparing the results presented in the literature, we implemented the algorithm for local alignments based on the chroma binary distance [3][2]. Given this dataset only has two recordings per song in the training set, notice that the maximum value for P@10 is 0.2.

TABLE III
MEAN AVERAGE PRECISION (MAP), PRECISION AT 10 (P@10), AND MEAN RANK OF FIRST CORRECTLY IDENTIFIED COVER (MR1) ON THE YOUTUBE COVERS DATASET

Algorithm	MAP	P@10	MR1
DTW	0.425	0.114	11.69
Silva et al. [12]	0.478	0.126	8.49
Serrà et al. [3]	0.525	0.132	9.43
SiMPle	**0.591**	**0.140**	**7.91**

Our method achieves the most accurate results in this experiment. Also, our method is notably faster than the second best (Serrà et al.). For a better understanding of the runtime, we tested all the algorithms used in this experiment. While

[2]Note that the algorithm proposed by Serrà et al. for cover song recognition uses 32 Harmonic Class Pitch Profile (HPCP) [20] features. In this work, we only used the proposed distance measure. Specifically, we compared the distances using the same set of features used by our method and DTW, i.e., CENS.

the original algorithm to calculate SiMPle is competitive with DTW and our new method can perform five times faster (c.f. Section III-B for details), the method proposed by Serrà et al. is more than ten times slower than DTW. We acknowledge that we did not prioritize optimizing the competing method. However, we do not believe that any code optimization is capable of significantly reducing the performance gap.

The presented results use 2 CENS per second and a subsequence representing 10 seconds. When we vary these parameters from the feature extraction and the SiMPle calculation phases, we obtain slightly different results.

Regarding the length of the subsequence, varying from 5 seconds to 60 seconds still provides us better results than the method proposed by Serrà et al. Specifically, using 15 seconds achieves slightly better results regarding MAP and MR1. In this case, we obtain 0.598 and 7.760 for these measures, respectively. The best P@10 is obtained when using 10 seconds.

On the other hand, the results presented more variability according to the feature extraction parameters. The previously presented results are the best obtained concerning the number of extracted features. When we double the number of features, the MAP, P@10, and MR1 obtained by SiMPle are 0.571, 0.132, and 8.68, respectively. If we use half of the features, these values are 0.549, 0.133, and 8.84. However, this change also affects all the assessed competitors. For instance, when we reduce the number of features by half, DTW achieves 0.3124, 0.0816, and 23.4960 for MAP, P@10, and MR1, respectively. Serrà's method achieves 0.4978, 0.1256, and 9.9600 for the same evaluation measures.

Finally, we also evaluated using different central tendency measures. Although we proposed the SiMPle-based distance using the median of local distances, we can use any other statistic. As previously stated, the median is more robust to outliers, such as insertions in the song that are not similar to any subsequence in its covers or its original recording. However, we noticed that using mean or geometric mean does not change the results significantly. In these cases, the differences in the assessed evaluation measures are in the order of 10^{-2}. On the other hand, the results obtained by using harmonic mean were slightly worse but still better than the adversary methods. Although we only report on varying the parameters for the YouTube Covers data, the conclusions are similar to the Mazurkas dataset.

Regarding the dataset of classical music, we report the results obtained by DTW and the MAP values documented in the literature. Notice that the values of P@10 and MR1 are not available for this dataset in the papers used for comparison. Specifically, the subset of Mazurkas used in this work is the same as the one used in [21] and [22] and has only minor differences with the dataset used in [23]. Although subtle variations of [3] are commonly used in state-of-the-art cover song identification systems [17], we do not include its results due to the high time complexity. Table IV shows the results.

In this dataset, most of the recordings conserve the structures of the pieces. In this case, DTW performs similarly to our algorithm. However, our method is approximately one order of magnitude faster, and it has several advantages over DTW such as an incremental property that allows our method to apply to data streams. Next section discusses such property.

TABLE IV
MEAN AVERAGE PRECISION (MAP), PRECISION AT 10 (P@10), AND MEAN RANK OF FIRST CORRECTLY IDENTIFIED COVER (MR1) ON THE MAZURKAS DATASET

Algorithm	MAP	P@10	MR1
DTW	**0.882**	0.949	4.05
Bello [21]	0.767	-	-
Silva et al. [22]	0.795	-	-
Grosche et al. [23]	0.525	-	-
SiMPle	0.880	**0.952**	**2.33**

D. *Streaming Cover Song Recognition*

Real-time audio matching has attracted attention in the last years. In this scenario, the input is a stream of audio, and the output is a sorted list of similar objects in a database.

In this section, we evaluate our algorithm in an online cover song recognition scenario. For concreteness, consider a TV station broadcasting a live concert. To automatically present the name of the song to the viewers or to synchronize the concert with a second screen app, we would like to take the streaming audio as input for our algorithm and recognize what song the band is playing as soon as possible. To accomplish this task, we need to match the input to a set of (previously processed) recordings.

In addition to allowing the fast calculation of all the distances of a subsequence to a whole song, the proposed algorithm has an incremental property that allows estimating cover song similarity in a streaming fashion. If we have a previously calculated SiMPle regarding the first few seconds of the song, then, when we extract a new vector of (chroma) features, we do not need to recalculate the whole SiMPle from the beginning. Instead, only two quick steps are required:

- First we calculate the distance profile to the new subsequence, i.e., the distance of the last observed subsequence (including the new feature vector) to all the subsequences of the original songs.
- Then it is necessary to update SiMPle by selecting the minimum value between the new distance profile and the previous SiMPle for each subsequence.

Notice that in the first step, we do not need to calculate the distance profile from scratch. Instead, we take advantage of the previously calculated values (c.f. Section II).

To evaluate the capability of our method for streaming recognition, we performed a simple experiment simulating the previously described scenario. First, we extracted features from each track in the Mazurkas dataset of original recordings. For clarity, we will refer to this database as the training set. Then, we randomly chose a "cover" recording as our query and processed it in the following manner. We began extracting features from the first two seconds of the query to calculate the first distance estimation to each training object. After this initial step, for each second of the query, we repeated the process of extracting features and re-estimating the distance measure to the training set.

In this experiment, we used 2 CENS per second. The training set is composed of one recording of each piece. We used a performance (which is not part of the training set) with approximately 92 seconds as a query, and the process was still faster than real-time. Specifically, the updates took approximately 0.28 second to extract the features, update SiMPle, and recalculate the distance to all the training objects for each second of "listened" performance.

Figure 6 demonstrates the changes in distance estimation in an audio streaming query session. In this case, we used a recording of the "Mazurka in F major, Op. 68, No. 3" as the query and a subsequence of two seconds. In the first distance estimation, i.e., after two seconds of audio, the correct class appears in the sixth position of the ranking. However, with more evidence, it quickly becomes the best match. Specifically, after three seconds of streaming, it goes to the fourth position. Four seconds from the beginning, the correct class is already considered the best match. It stands until the streaming ends.

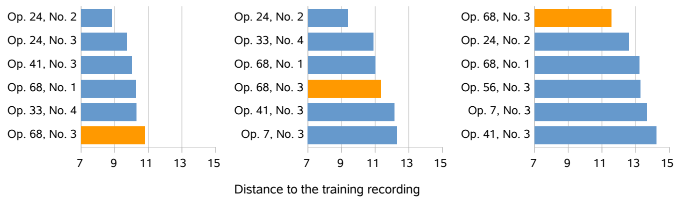


Fig. 6. Changes in the distance when querying a recording of the "Mazurka in F major, Op. 68, No. 3" in a streaming fashion. The graphs represent the top five matches after processing two (*left*), three (*middle*), and four (*right*) seconds of the audio.

We experimented to verify how fast is the streaming cover recognition. For this purpose, we randomly chose five versions of each of the 49 pieces in the Mazurkas dataset and calculated the time taken for the version to reach the third and the first position in the similarity ranking. We did not use all the versions to avoid pieces with a high number of versions to dominate the results.

For some recordings, our method is not able to rank the correct version as the best match. Although these are a few cases, they may significantly affect the runtime statistics. For this reason, we assessed the time to achieve the top positions in the ranking using two different sets of results. The first only considers the cases in which the algorithm identified the original version in these positions. The second one considers all the cases.

Table V presents the mean and median time in these evaluation scenarios. We compute both centrality measures to account for the variability in the times necessary to identify the cover song.

In our experimental design, the first rank assessment occurs after two seconds of music execution. In more than half of the recordings in the subset, our algorithm correctly ranks the version among the top 3 right in the first iteration. When we consider all cases, our method needs to listen for 4 seconds to rank 50% of the versions in the top 3. Regarding the first position in the ranking, we need 6 and 39 seconds in these scenarios, respectively.

TABLE V
TIME TO ACHIEVE THE TOP 1 AND TOP 3 POSITIONS IN THE SIMILARITY RANKING IN THE STREAMING SCENARIO. THE "AVERAGE TIME - SUBSET" AND "MEDIAN TIME - SUBSET" CONSIDER THE SUBSET OF CASES OUR ALGORITHM RANKS THE CORRECT VERSION IN THE TOP 1 AND 3. "MEDIAN TIME - ALL" CONSIDERS ALL CASES, INCLUDING THE ONES OUR PROPOSAL DOES NOT CORRECTLY RANK THE VERSIONS AMONG THE TOP 3.

	Top 1	Top 3
Average time - subset	17.11s	8.93s
Median time - subset	6.00s	3.00s
Median time - all	39.00s	4.00s

E. SiMPle-Based Audio Thumbnailing

Audio thumbnails are short representative excerpts of audio recordings. Thumbnails have several applications in music information retrieval. For example, they can be used to show the result of a search to the user. In a commercial application, they can be used as the preview to a potential customer in an online music store.

There is a consensus in the music information retrieval community that the "ideal" music thumbnail is the most repeated excerpt, such as the chorus [24]. By using this assumption, the application of SiMPle to identify a thumbnail is direct. Consider the SiMPle index obtained by the self-join procedure. The thumbnail is the subsequence which is the nearest neighbor of other subsequences most times. In other words, the mode of the SiMPle index points to the beginning of the thumbnail.

This method, first proposed by [9], is simple and completely fits the assumption of the thumbnail be the most repeated excerpt. However, we noticed that it is very susceptible to draws. In other words, there are many recordings with more than one mode in the SiMPle index.

To testify this fact, we calculated the SiMPle and SiMPle index of each track in the YouTube Covers data and counted the number of tied cases. When using 2 CENS per second to identify 30-seconds thumbnails, 197 from 350 examples presented draws. Considering all the recordings, the average number of draws per song is 4.15. When limiting this calculation to the 197 cases containing draws, the average is 7.36. Clearly, if we use fewer features per second, we will have a lower number of subsequences to be assessed and, consequently, the number of draws tends to decrease. On the other hand, if we use a higher number of features per second, the number of examples with tied subsequences tends to increase.

In the original proposal, in the case of drawing, the algorithm took the first tied subsequence as the thumbnail. We propose to use the values in the SiMPle to untie these cases. The pair of nearest subsequences with smaller distance is, intuitively, the most "faithful" repetition in the time series. Similarly, the tied subsequence that is, on average, the most similar to its neighbors is the most faithfully repeated one.

For this purpose, while calculating the mode, we also store the summed distances between the neighbors. In other words, while we use the SiMPle index to count the number of times each subsequence is the nearest neighbor of others, we sum the values stored in the SiMPle in these cases. In the end, we take the subsequences the maximum number of times in the SiMPle index pointed as candidates for the thumbnail. If this operation returns only one subsequence, this is considered the thumbnail. Otherwise, we assume the candidate with the smallest sum of distances as the thumbnail of that recording.

At this point, the reader must be wondering how effective is our untying strategy. To answer this question, we compared the original and the proposed methods. Regarding the 197 tied recordings in the YouTube Covers data, the new algorithm differs from the original in 130 cases. It represents 66% of the cases.

A commonly observed case of draw occurs when some subsequences comprise pre-chorus excerpts. In this case, the chorus and the pre-chorus contain the same number of neighbors pointed in the SiMPle index. However, the pre-chorus commonly present slight differences caused by some word substitution or distinct intonation by the singer, which induces a subtle difference in the chroma features. This effect is usually much lower in the chorus.

Figure 7 shows one example of this fact. It represents a histogram of the SiMPle index obtained from the song "Like a Rolling Stone" by Bob Dylan using a subsequence representing 30 seconds. We can notice many tied cases. Specifically, the first one occurs at 81 seconds of the song and comprises part of the first pre-chorus and a small portion of the chorus. It presents an average distance of 13.11 from its neighbors, while there is another tied subsequence much smaller average distance. This other subsequence is on average 6.59 units far from their neighbors. The proposed untying algorithm, then, returns this last subsequence, which starts at 153 seconds, comprising exactly the chorus of the song.

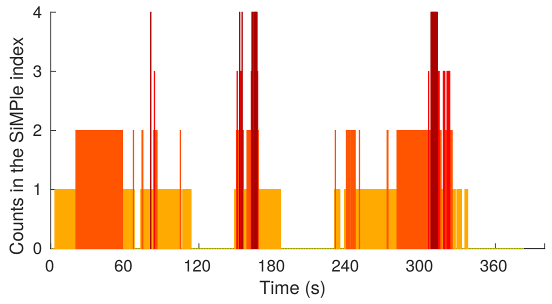


Fig. 7. Histogram of the SiMPle index for the song "Like a Rolling Stone." Each bar counts how many times the subsequence started at that point was considered the nearest neighbor of any other. The first prominent peak (starting at 81 seconds of the song) comprises pieces of a pre-chorus and a chorus. However, the proposed untying algorithm selects a subsequence which comprises only the chorus (starting at 153 seconds).

The proposed method is simple and fast. If the SiMPle and SiMPle index are already calculated for other taks, we only need an additional $\mathcal{O}(n)$ procedure to find a thumbnail in it. Otherwise, both operations, counting and summing distances, can be performed during the computation of the SiMPle and the SiMPle index. In this case, these additional operations do not affect the runtime of SiMPle-Fast significantly.

IV. Conclusion

In this paper, we introduced a technique to exploit subsequences joins to assess similarity in music. The presented method is very fast and requires only one parameter, the subsequence length, that is intuitively set in music applications. Moreover, we showed that our method is robust for the choice of this parameter for a range of values.

While we focused our evaluation on the cover song recognition and audio thumbnailing, our approach has the potential for applications in different MIR tasks. We intend to further investigate the use of matrix profiles in other tasks and the effects of different features in the process.

One limitation of the proposed method is that the use of only one nearest neighbor is sensitive to hubs, i.e., subsequences considered the nearest neighbor of many other snippets. For instance, a hub subsequence in an original recording can severely affect the cover song recognition performance. In this case, the recording containing this fragment tends to appear in the top positions for a multitude of queries, cover or not. Better understanding this and other effects of analyzing subsequences similarity join in music data is also part of our future work.

Despite the fact that our method is much faster than the previously proposed algorithm, it still faces difficulties when scaling to massive datasets. When using SiMPle for similarity-based recovery tasks, such as cover song recognition, indexing the search procedure usually provides a significant speedup. Given that our method is based on the Euclidean distance, we believe that we can adapt some methods for indexing in metric space to speed up the search based on SiMPle. As another direction for future work, we intend to seek possible solutions for this issue.

We notice that we are committed to the reproducibility of our results, and we encourage researchers and practitioners to extend our ideas and evaluate using the SiMPle in different MIR tasks. To this end, we have created a website[3] with the complete source code used in our experiments and videos highlighting some of the results presented in this work.

Acknowledgments

This work is an expanded version of the paper that appeared in ISMIR 2016 [9], in which we proposed the SiMPle for music analysis. In this expanded version, we adapted the algorithm to calculate our representation to a much faster method, which utilizes previous calculations to reduce the number of computations. Also, we extended the cover song recognition by experimenting its robustness against structural variations and the effectiveness of streaming version and the audio thumbnailing by proposing a SiMPle-based untying method and discussing its relevance and its contribution.

This work was funded by grants #2013/26151-5 and #2016/04986-6 São Paulo Research Foundation (FAPESP), 446330/2014-0 and 306631/2016-4 Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), and IIS-1161997 II National Science Foundation (NSF).

References

[1] J. Serra, M. Müller, P. Grosche, and J. L. Arcos, "Unsupervised music structure annotation by time series structure features and segment similarity," *IEEE Transactions on Multimedia*, vol. 16, no. 5, pp. 1229–1240, 2014.

[3]https://sites.google.com/view/simple-fast

[2] S. Wang, S. Ewert, and S. Dixon, "Robust and efficient joint alignment of multiple musical performances," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 11, pp. 2132–2145, 2016.

[3] J. Serra, E. Gómez, P. Herrera, and X. Serra, "Chroma binary similarity and local alignment applied to cover song identification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 6, pp. 1138–1151, 2008.

[4] N.-H. Liu, "Effective results ranking for mobile query by singing/humming using a hybrid recommendation mechanism," *IEEE Transactions on Multimedia*, vol. 16, no. 5, pp. 1407–1420, 2014.

[5] T. Li, M. Ogihara, W. Peng, B. Shao, and S. Zhu, "Music clustering with features from different information sources," *IEEE Transactions on Multimedia*, vol. 11, no. 3, pp. 477–485, 2009.

[6] H.-H. Wu and J. P. Bello, "Audio-based music visualization for music structure analysis," in *Proceedings of Sound and Music Computing Conference (SMC)*, 2010.

[7] J. Foote, "Visualizing music and audio using self-similarity," in *ACM International Conference on Multimedia*. ACM, 1999, pp. 77–80.

[8] J. Serra, X. Serra, and R. G. Andrzejak, "Cross recurrence quantification for cover song identification," *New Journal of Physics*, vol. 11, no. 9, p. 093017, 2009.

[9] D. F. Silva, C.-C. M. Yeh, G. E. A. P. A. Batista, and E. Keogh, "SiMPle: assessing music similarity using subsequences joins," in *International Society for Music Information Retrieval Conference*, 2016, pp. 23–29.

[10] A. Mueen, K. Viswanathan, C. Gupta, and E. Keogh, "The fastest similarity search algorithm for time series subsequences under euclidean distance," www.cs.unm.edu/~mueen/FastestSimilaritySearch.html, accessed 24 Apr, 2017.

[11] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Matrix Profile II: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins," in *IEEE International Conference on Data Mining*. IEEE, 2016, pp. 739–748.

[12] D. F. Silva, V. M. A. d. Souza, G. E. A. P. A. Batista *et al.*, "Music shapelets for fast cover song regognition," in *International Society for Music Information Retrieval Conference*, 2015, pp. 441–447.

[13] C. Sapp, "The mazurka project," http://www.mazurka.org.uk, accessed 24 Apr, 2017.

[14] M. Müller, F. Kurth, and M. Clausen, "Audio matching via chroma-based statistical features," in *International Society for Music Information Retrieval Conference*, pp. 288–295.

[15] M. Muller, "Dynamic time warping (dtw)," *Information Retrieval for Music and Motion*, pp. 70–83, 2007.

[16] W.-H. Tsai, H.-M. Yu, and H.-M. Wang, "Using the similarity of main melodies to identify cover versions of popular songs for music document retrieval." *J. Inf. Sci. Eng.*, vol. 24, no. 6, pp. 1669–1687, 2008.

[17] N. Chen, W. Li, and H. Xiao, "Fusing similarity functions for cover song identification," *Multimedia Tools and Applications*, pp. 1–24, 2017.

[18] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[19] J. Serra, E. Gómez, and P. Herrera, "Transposing chroma representations to a common key," in *IEEE CS Conference on The Use of Symbols to Represent Music and Multimedia Objects*, 2008, pp. 45–48.

[20] E. Gómez, "Tonal description of music audio signals," *Department of Information and Communication Technologies*, 2006.

[21] J. P. Bello, "Measuring structural similarity in music," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 7, pp. 2013–2025, 2011.

[22] D. F. Silva, H. Papadopoulos, G. E. A. P. A. Batista, and D. P. W. Ellis, "A video compression-based approach to measure music structural similarity," in *14th International Society for Music Information Retrieval Conference (ISMIR 2014)*, 2013, pp. 95–100.

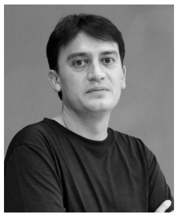
[23] P. Grosche, J. Serra, M. Müller, and J. L. Arcos, "Structure-based audio fingerprinting for music retrieval," in *International Society for Music Information Retrieval Conference*, 2012, pp. 55–60.

[24] M. A. Bartsch and G. H. Wakefield, "Audio thumbnailing of popular music using chroma-based representations," *IEEE Transactions on Multimedia*, vol. 7, no. 1, pp. 96–104, 2005.

Diego F. Silva concluded his Ph.D. in 2017 in Computer Science and Computational Mathematics at the Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Brazil, with a one-year research internship at the University of California Riverside. Now, he is an assistant professor at the Universidade Federal de São Carlos, Brazil. He is author of papers in relevant conferences and journals. His research interests include data mining and music information retrieval.

Chin-Chia M. Yeh received his B.S. and M.S. degree from Virginia Tech and University of California, Los Angeles in 2010 and 2011, respectively. From 2011 to 2014, he joined Research Center for Information Technology Innovation at Academia Sinica, Taiwan as a Research Assistant. He has been a Ph.D Student at University of California, Riverside since 2014. His research interests include time series analysis, data mining, and machine learning.

Yan Zhu is a fourth year Ph.D. student at the University of California, Riverside. She obtained her M.S. and B.S. degrees in 2013 and 2010, respectively, both from Shanghai Jiao Tong University, Shanghai, China. She has published papers in top data mining venues, including ICDM, ECML-PKDD, Data Mining and Knowledge Discovery, etc. Her research interests include data mining and machine learning.



Gustavo E. A. P. A. Batista Gustavo E. A. P. A. Batista is an associate professor in Computer Science at Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Brazil. He earned his Ph.D. in Computer Science (2003) and was a visiting researcher at University of California, Riverside (2010-2011). His research interests include machine learning, data mining and time series processing with applications in Music Retrieval and Entomology. He is author of more than 100 peer-reviewed papers in conferences and journals.

Eamonn Keogh Dr. Keogh is a full professor at the University of California Riverside. He is a prolific author in data mining, a top-ten most prolific author in all three of the top ranked data mining conferences, SIGKDD, ICDM and SDM. He has won best paper awards at ICDM, SIGKDD and SIGMOD, and he has given well received tutorials at SIGKDD (five times), ICDM (four times), VLDB, SDM, and CIKM.