SIMPLE: ASSESSING MUSIC SIMILARITY USING SUBSEQUENCES JOINS

Diego F. Silva[1,2], Chin-Chia M. Yeh[2],Gustavo E. A. P. A. Batista[1], Eamonn Keogh[2]
[1] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Brazil
[2] Department of Computer Science and Engineering, University of California, Riverside, USA
diegofsilva@icmc.usp.br,myeh003@ucr.edu,gbatista@icmc.usp.br,eamonn@ucr.edu

ABSTRACT

Most algorithms for music information retrieval are based on the analysis of the similarity between feature sets extracted from the raw audio. A common approach to assessing similarities within or between recordings is by creating similarity matrices. However, this approach requires quadratic space for each comparison and typically requires a costly post-processing of the matrix. In this work, we propose a simple and efficient representation based on a subsequence similarity join, which may be used in several music information retrieval tasks. We apply our method to the cover song recognition problem and demonstrate that it is superior to state-of-the-art algorithms. In addition, we demonstrate how the proposed representation can be exploited for multiple applications in music processing.

1. INTRODUCTION

With the growing interest in applications related to music processing, the area of music information retrieval (MIR) has attracted huge attention in both academia and industry. However, the analysis of audio recordings remains a significant challenge. Most algorithms for content-based music retrieval have at their cores some similarity or distance function. For this reason, a wide range of applications rely on some technique to assess the similarity between music objects. Such applications include segmentation [8], audio-to-score alignment [4], cover song recognition [15], and visualization [23].

A common approach to assessing similarity in music recordings is achieved by utilizing a self-similarity matrix (SSM) [5]. This representation reveals the relationship between each "snippet" 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, as in the application of cross-recurrence analysis for cover song recognition [16].

The main advantage of similarity matrices is the fact that they simultaneously reveal both the *global* and the *local* structure of music recordings. However, this representation requires quadratic space in relation to the length of the feature vector used to describe the audio. For this reason, most methods to find patterns in the similarity matrix are (at least) quadratic in time complexity. In spite of this, most information contained in similarity matrices is irrelevant or has little impact in its analysis. This observation suggests the need for a more space and time efficient representation of music recordings.

In this work, we extend the *subsequences all-pairs-similarity-search*, also known as *similarity join*, in order to assess the similarity between audio recordings for MIR tasks. As with the common similarity matrices, representing the entire subsequence join requires a quadratic space, and also has a high time complexity, which is dependent on the length of the subsequences to be joined.

However, in this work we show that we can exploit a new data structure called *matrix profile* which allows a space efficient representation of the similarity join matrix between subsequences. Moreover, we can leverage recent optimizations in FFT-based all-neighbor search that allow the matrix profile to be computed efficiently [10]. For clarity, we refer to the representation presented in this paper as Similarity Matrix ProfiLE (SiMPle).

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

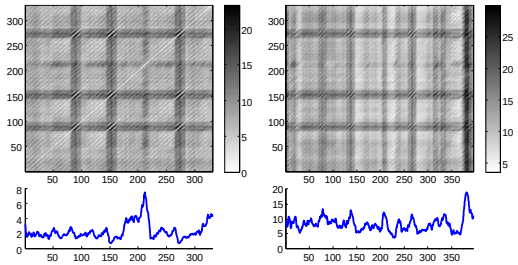

Figure 1. Similarity matrix within (*left*) and between different songs (*right*) and their respective SiMPle.

In summary, our method has the following advantages/features:

- It is a novel approach to assess the audio similarity and can be used in several MIR algorithms;
- We exploit the fastest known subsequence similarity search technique in the literature [10], which makes our method fast and exact;
- It is simple and only requires a single parameter, which is intuitive to set for MIR applications;
- It is space efficient, requiring the storage of only $O(n)$ values;
- Once we calculate the similarity profile for a dataset it can be efficiently updated, which has implications for streaming audio processing.

2. SiMPle: SIMILARITY MATRIX PROFILE

We begin by describing the operation for producing the matrix profile, a *similarity join*. For clarity, we use the term *time series* to refer to the ordered set of features that describe a whole recording and *subsequence* to define any continuous subset of features from the time series.

Definition 1: *Similarity join*: given two time series *A* and *B* with the desired subsequence length *m*, the similarity join identifies the nearest neighbor of each subsequence (with length *m*) in *A* from all the possible subsequence set of *B*.

Through such a similarity join, we can gather two pieces of information about each subsequence in *A*, which are: 1) the Euclidean distance to its nearest neighbor in *B* and 2) the position of its nearest neighbor in *B*. Such information can be compactly stored in vectors, referred as *similarity matrix profile* (SiMPle) and similarity *matrix profile index* (SiMPle index) respectively.

One special case of similarity join is when both input time series refer to the same recording. We define the operation that handles this specific case *self-similarity join*.

Definition 2: *Self-similarity join*: given a time series *A* with the desired subsequence length *m*, the self-similarity join identifies the non-trivial nearest neighbor of each subsequence (with length *m*) in *A* from all the possible subsequence set of *A*.

The only major difference between self-similarity join (**Definition 2**) and similarity join (**Definition 1**) is the exclusion of trivial matched pairs when identifying the nearest neighbor. The exclusion of trivial matches is crucial as matching a subsequence with itself (or slightly shifted version of itself) produces no useful information.

We describe our method to calculate SiMPle in Algorithm 1. In line 1, we record the length of *B*. In line 2, we allocate memory and initialize SiMPle P_{AB} and SiMPle index I_{AB}. From line 3 to line 6, we calculate the *distance profile* vector *D* which contains the distances between a given subsequence in time series *B* and each subsequence in time series *A*. The particular function we used to compute *D* is *MASS* (Mueen's Algorithm for Similarity Search), which is the most efficient algorithm known for distance vector computation [10]. We then perform the pairwise minimum for each element in *D* with the paired element in P_{AB} (i.e., min($D[i]$, $P_{AB}[i]$) for $i = 0$ to length(D) - 1.) We also update $I_{AB}[i]$ with *idx* when $D[i] \leq P_{AB}[i]$ as we perform the pairwise minimum operation. Finally, we return the result P_{AB} and I_{AB} in line 7.

Algorithm 1. Procedure to calculate SiMPle and SiMPle index

Input: Two user provided time series, *A* and *B*, and the desired subsequence length *m*
Output: The SiMPle P_{AB} and the associated SiMPle index I_{AB}

```
1  n_B ← Length(B)
2  P_AB ← infs, I_AB ← zeros, idxes ← 1:n_B-m+1
3  for each idx in idxes
4        D ← MASS(B[idx:idx+m-1], T_A) // c.f. [10]
5        P_AB, I_AB ← ElementWiseMin(P_AB, I_AB, D, idx)
6  end for
7  return P_AB, I_AB
```

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 lines 1 and 4 and ignore trivial matches in *D* when performing *ElementWiseMin* in line 5.

The method *MASS* (used in line 4) is important to speed-up the similarity calculations. This algorithm has a time complexity of $O(n \log n)$. For brevity, we refer the reader interested in details of this method to [10].

In this work, we focus on demonstrating the utility of SiMPle on the cover song recognition task. Given that the cover song recognition is a specialization of the "query-by-similarity" task, we believe that it is the best scenario to evaluate a similarity method. Specifically, we propose a SiMPle-based distance measure between a query and its potential original version.

3. 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 [20] or locally [15][18] conserved structure(s). A well-known and widely applied algorithm for measuring the global similarity between tracks is Dynamic Time Warping (DTW) [11]. In spite of its utility in other domains, DTW is not generally robust to 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 boundaries detection can cause a domino effect (compounded errors) in the whole identification process.

In addition, the complexity of the algorithm to calculate DTW is $O(n^2)$. Although methods to fast approximate the DTW have been proposed [13], there is no error bound for such approximations. In other words, it is not possible to set a maximum error in the value obtained by it in relation 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 widely used method for music similarity proposes the use of a binary distance function to compare chroma-based features followed by a dynamic programming local alignment [15]. Despite its demonstrated utility to recognize cover recordings, this method has several parameters, that are unintuitive to tune, and is slow. 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.

3.1 SiMPle-Based Cover Song Recognition

In this work, we propose to use SiMPle to measure the distance between recordings in order to identify cover songs. In essence we exploit the fact that the *global* relation between the tracks is composed of many *local* simi-

larities. In this way, we are able to simultaneously take advantage of both local and global pattern matching.

Intuitively, we should 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 1.

$$dist(A,B)=median(SiMPle(B,A)) \quad (1)$$

Note that several other measures of statistics could be used instead of the median. However, the median is robust to outliers in the matrix profile. Such 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 to this situation, as well as other changes in structure, is discussed in the next section.

3.2 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 of 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 feature is *inserted* into a cover, this will have as consequence a peak in the SiMPle that will cause only a slight increase in its median value.

4. EXPERIMENTAL EVALUATION

The evaluation of different choices of features sets is not the main focus of this paper. For this reason, we fix the use of chroma-based features in our experiments, as it is the most popular feature set to analyze music data. In order to provide local tempo invariance, we used the chroma energy normalized statistics (CENS) [12]. Specifically, for the cover song recognition task, we adopted the rate of two CENS per second of audio.

In addition, we preprocessed the feature sets in each comparison to provide key invariance. Before calculating the similarity between songs, we transpose one of them in order to have the same key using the optimal transposition index (OTI) [14].

We notice that we are committed to the reproducibility of our results, and we encourage researchers and practitioners to extend our ideas and evaluate the use of the SiMPle in different MIR tasks. To this end, we created a website [19] with the complete source code used in our experiments and videos highlighting some of the results presented in this work.

4.1 Datasets

We evaluate our method in different scenarios regarding music styles and size of the databases. Specifically, we tested the proposed distance measure's utility for assessing both popular and classical recordings.

The first database considered is the YouTube Covers [18], composed of 50 different compositions, each one containing 7 different recordings obtained from YouTube videos. The data was originally split into training and testing partitions, in which the training set is composed of the original recording in studio and a live version performed by the same artist. To allow comparisons to the literature, we follow the same configuration.

The second dataset we consider is the widely used collection of Chopin's Mazurkas. 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.

4.2 Results and Discussion

In order to assess the performance of our method, we used three commonly applied evaluation measures: mean average precision (MAP), precision at 10 (P@10), and mean rank of first correctly identified cover (MR1). Note that for MR1, smaller values are better.

For both the YouTube Covers and Mazurkas datasets, we compared our algorithm using results previously presented in the literature. For the former case, in addition to comparing to the results presented in the paper for which the dataset was created [18], we carefully implemented the algorithm for local alignments based on the chroma binary distance [15]. Table 1 shows the results.

Algorithm	MAP	P@10	MR1
DTW	0.425	0.114	11.69
Silva et al. [18]	0.478	0.126	8.49
Serrà et al. [15]	0.525	0.132	9.43
SiMPle	**0.591**	**0.140**	**7.91**

Table 1. Mean average precision (MAP), precision at 10 (P@10), and mean rank of first correctly identified cover (MR1) on the YouTube Covers dataset. Given that this dataset has only two recordings per song in the training set, the maximum value to P@10 is 0.2.

Our method achieved the best results in this experiment. In addition, we note that our method is notably faster than the second best (Serrà et al.). Specifically, while our method took 1.3 hours, the other method took approximately one week to run on the same computer[1].

[1] In our experiments, we used an 8-core Intel® Core ™ i7-6700K CPU @ 4.00GHz with 32GB of RAM memory running Windows 10®. All our codes were implemented and executed using Matlab R2014a®.

We acknowledge that we did not invest a lot of effort optimizing the competing method. However, we do not believe that any code optimization is capable of significantly reducing the performance gap.

We also consider the Mazurkas dataset. In addition to the results achieved by DTW, we report MAP results documented in the literature, which were achieved by retrieving the recordings by structural similarity strategies using this data. Specifically, the subset of mazurkas used in this work is exactly the same as the used in [2] and [17] and has only minor differences to the dataset used in [6]. Although [15] is considered the state-of-the-art for cover song recognition, we do not include its results due to the high time complexity. Table 2 shows the results.

Algorithm	MAP	P@10	MR1
DTW	**0.882**	0.949	4.05
Bello [2]	0.767	-	-
Silva et al. [17]	0.795	-	-
Grosche et al. [6]	0.819	-	-
SiMPle	0.880	**0.952**	**2.33**

Table 2. Mean average precision (MAP), precision at 10 (P@10), and mean rank of first correctly identified cover (MR1) on the Mazurkas dataset.

The structures of the pieces on this dataset are respected in most of the recordings. In this case, DTW performs similar than our algorithm. However, our method is faster (approximately two times in our experiments) and has several advantages over DTW, such as its incremental property, discussed in the next section.

4.3 Streaming Cover Song Recognition

Real-time audio matching has attracted the attention of the community 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 that a TV station is broadcasting a live concert. In order 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 to our algorithm and be able to 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 can be exploited to estimate cover song similarity in a streaming fashion. If we have a previously calculated SiMPle, then, when we extract a new vector of (chroma) features, we do not need to recalculate the whole SiMPle from the beginning. Instead, just two quick steps are required:

- Calculation of 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 song;
- Update of SiMPle by selecting the minimum value between the new distance profile and the previous SiMPle for each subsequence.

These steps are done by the Algorithm 2.

Algorithm 2. Procedure to incrementally update SiMPle and SiMPle index

Input: The current time series A and B, the new chroma vector c, the desired subsequence length m, and the current SiMPle P_{AB} and SiMPle index I_{AB}

Output: The updated SiMPle $P_{AB,new}$ and the associated SiMPle index $I_{AB,new}$

1 $newB \leftarrow$ Concatenate(B,c), $n_B \leftarrow$ Length($newB$)
2 $D \leftarrow$ MASS($newB[n_B\text{-}m\text{+}1: n_B]$, A) // c.f. [10]
3 P_{AB}, $I_{AB} \leftarrow$ ElementWiseMin(P_{AB}, I_{AB}, D, $n_B\text{-}m\text{+}1$)
4 $P_{AB,last}$, $I_{AB,last} \leftarrow$ FindMin(D)
5 $P_{AB,new} \leftarrow [P_{AB}, P_{AB,last}]$, $I_{AB,new} \leftarrow [I_{AB}, I_{AB,last}]$
6 **return** $P_{AB,new}$, $I_{AB,new}$

To evaluate the ability 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 dataset of original recordings. For clarity, we will refer to this database as the *training set*. Then, we randomly chose another recording as our query and processed it according to the following steps. We begin extracting features from the first three seconds of the query in order to calculate the first distance estimation to each training object. After this initial step, for each second of the query, we repeat the process of extracting features and re-estimating the distance measure to the training set.

In this experiment, we used the Mazurkas dataset with two CENS per second. The training set is composed of the first recording (in alphabetical order) of each piece. We used a performance with approximately 275 seconds as a query, and we were able to maintain the process faster than real-time. Specifically, the updates took approximately 0.7 seconds to extract the features, update SiMPle, and recalculate the distance for all the training objects.

Figure 2 visualizes 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 query. In the first estimation, its training version appears as the sixth nearest neighbor. However as we see more evidence, it quickly becomes the best match.

Figure 2. 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 6 matches after processing 3 (*left*), 5 (*middle*), and 10 (*right*) seconds of the audio.

Another strategy that can be used in this scenario is an amnesic sliding window, in order to forget old values and further speedup the matching of new subsequences. For a given window length *w*, we can maintain just the last *w* values in the SiMPle. In this way, a change in the distribution of the distance estimates may assist in the identification of the ending and beginning of a song. At the same time, the positions of the most recently matched sections can be used as estimation of the moment in the current song. These ideas may help to identify songs being sequentially played in a random or unknown order.

5. EXPANDING THE RANGE OF APPLICATIONS OF SiMPle

In this work, we focus on assessing music similarity by joining subsequences. While we evaluate our method on the cover song recognition task, we claim that the SiMPle is a powerful tool for other music processing tasks. To reinforce this argument, we present insights on how to use SiMPle in different application domains, as well some initial results. The methods presented in this section have room for improvements, but they are simple yet effective. We intend to further explore and evaluate SiMPle in (at least) the tasks listed below.

In contrast to the previous experiments, when we use the self-similarity join to highlight points of interest in a recording, we apply ten CENS per second.

5.1 Motifs and Discords

The SiMPle from a self-similarity join has several exploitable properties. For example, the lowest points correspond to the locations of the most faithfully repeated section (i.e., the *chorus* or *refrain*). Between several definitions of *motifs* in the literature, such as *harmonic* or *melodic motifs*, its simplest definition is the *closest pair of subsequences*. As noted in the time series literature, given the best motif pair, other definitions of motifs can be solved by minor additional calculations [9].

On the other hand, the highest point on the SiMPle corresponds to the "most unique" snippet from the recording. The procedure to search for such a subsequence that is the furthest from any other, known as *discord discovery*, can be used in music processing to find interesting segments in recordings. For example, it can be used to identify a solo, improvisation segments or the bridge.

For example, consider the song "*Let It Be*" by The Beatles. Figure 3. shows the SiMPle obtained for this track and points to their discord and pair of best motifs.

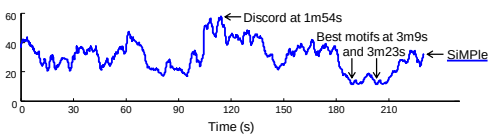

Figure 3. The pair of best motifs in a recording is determined by the subsequences starting at the positions of the minimum values of its SiMPle. At the same time, the position of the highest value points to the beginning of the discord excerpt.

While the motifs point to refrains, the discord includes bridge and the beginning of the guitar solo.

5.2 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 as the snippet shown the result of a search to the user. In a commercial application, they can be used as the preview to a potential costumer in an online music store.

There is a consensus in the MIR community that the "ideal" music thumbnail is the most repeated excerpt, such as the chorus [1]. 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 given by the subsequence starting in the position that is most used as a nearest neighbor. In other words, the beginning of the thumbnail is given by the position related to the mode of SiMPle index.

To illustrate this idea, we considered the song "*New York, New York"* by Frank Sinatra. Looking for a 30 seconds thumbnail, we found an excerpt that is comprised of the last refrain, as well as the famous (brass) instrumental basis of the song. Figure 4 shows the histogram of the SiMPle index found in this experiment.

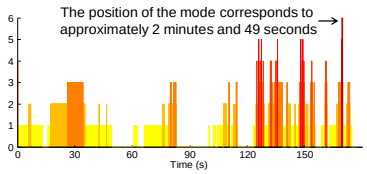

Figure 4. Histogram of SiMPle index for the song "*New York, New York*". Each bar counts how many times the subsequence starting at that point was considered the nearest neighbor of any other. We consider the subsequence represented by the most prominent peak as the thumbnail for this recording.

5.3 Visualization

The visualization of music structure aids the understanding of the music content. Introduced in [21], the arc diagram is a powerful tool to visualize repetitive segments in MIDI files [22] and audio recordings [23]. This approach represents a song by plotting arcs linking *repeated* segments.

All the information required to create such arcs are completely comprised on the SiMPle and the SiMPle index obtained by a self-join. Specifically, SiMPle provides the distances between subsequences, which can be used to determine if they are similar enough to exist a link between them and to define the color or transparency of each arc. The SiMPle index can be used to define both the positions and width of the arcs.

Figure 5 shows the scatter plot of the SiMPle index for "*Hotel California*" by Eagles. In this figure, there is a point (x,y) only if y is the nearest neighbor of x. The clear diagonals on this plot represent regions of n points such that the nearest neighbors of $[x,x+1,\ldots,x+n-1]$ are approximately $[y,y+1,\ldots,y+n-1]$. If the distance between such excerpts is low, then these regions may have a link between them. For this example, we defined the mean value of the SiMPle in that region as the distance threshold between the segments, in order to resolve if they should

stablish a link. Such threshold has direct impact on the number of arcs plotted.

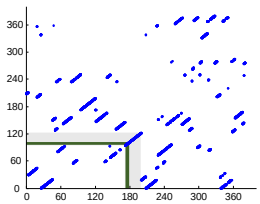

Figure 5. Scatter plot of the SiMPle index for the song "*Hotel California*". The (light) gray area indicates a possible link, but only the values in the (dark) green area represent subsequences with distance lower than the threshold.

By using a simple algorithm to spot such diagonals, we only need to define a threshold of distance and minimum length of the linkages. We set the width of the links in our experiment to be greater than or equal to 5 seconds. Figure 6 shows the resulting arc plot for the example shown in Figure 5.

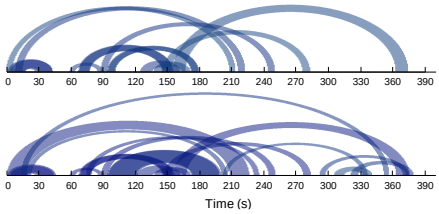

Figure 6. Arc plot for the song "*Hotel California*". These plots show the difference between using the mean value of SiMPle as distance threshold (*above*) and no distance threshold at all (*below*). The color of the arcs are related to their relevance, i.e., as darker the arc, closer the subsequences linked by it.

5.4 Endless Reproduction

Consider a music excerpt s_1, which starts at the time t_1 of a specific song, has a small distance to its nearest neighbor s_2, which starts at time t_2. When the reproduction of this song arrives t_1, we can make a random decision to "skip" the reproduction to t_2. Given that s_1 and s_2 are similar, this jump may be imperceptible to the listener. By creating several points of skip, we are able to define a sequence of jumps that creates an endless reproduction of the song. A well-known deployed example of this kind of player is the Infinite Jukebox [7].

The distance values obtained by the self-join represent how similar each subsequence is to its nearest neighbor in another region of the song. Adopting a small threshold to the distance between subsequences, we can use SiMPle to define the jumps. These characteristics may be explored in order to create a player for endless reproduction. We refer the interested reader to the supporting website [19] for examples of this functionality.

5.5 Sampling Identification

In addition to providing a *global* distance estimation between different songs, SiMPle is also powerful to examine *local* similarities. An interesting application that may exploit this ability is the automatic identification of samples. Sampling is the act of "borrowing" the instrumental basis or main melody from another song. This is a common approach in electronic and hip-hop music.

In contrast to cover versions, sampling is used as a virtual "instrument" to compose new songs. However, algorithms that look only for local patterns to identify versions of the same track may classify a recording using samples as a cover song. Using SiMPle, we can discover that the sampling excerpts have small distance values. In contrast, the segments related to the new song have significantly higher values.

Figure 7 shows an example of the usage of SiMPle to spot sampling. In this case, we compare the song "*Under Pressure*" by Queen and David Bowie with "*Ice Ice Baby*" by Vanilla Ice. Most of the continuous regions with values lower than the mean refer to the sampling of the famous bass line of the former song.

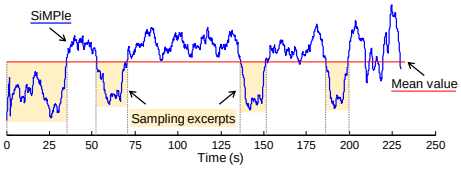

Figure 7. SiMPle (in blue) obtained between the songs "*Ice Ice Baby*" and "*Under Pressure*". The continuous regions below the mean value (in red) represent the excerpts sampled by Vanilla Ice from Queen's song.

6. CONCLUSIONS

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 that is intuitively set in music applications.

While we focused our evaluation on the cover song recognition, we have shown that our approach has the potential for applications in different MIR tasks. We intend to further investigate the use of matrix profiles in the tasks discussed in Section 5 and the effects of different features in the process.

The main limitation of the proposed method is that the use of only one nearest neighbor may be sensitive to hubs, i.e., subsequences that are considered the nearest neighbor of many other snippets. In addition, SiMPle cannot be directly used to identify regions where several subsequences are next to each other, composing a dense region. For this reason, we intend to measure the impact of the reduction in the amount of information in different tasks. Given that, we plan to explore how to incorporate additional information to SiMPle with no loss of time and space efficiency.

We have encouraged the community to confirm our results and explore or extend our ideas by making the code freely available [19].

Acknowledgements: The authors would like to thank FAPESP by the grants #2013/26151-5 and #2015/07628-0 and CNPq by the grants #303083/2013-1 and #446330/2014-0, and NSF by the grant IIS 1510741.

7. REFERENCES

[1] 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.

[2] J. P. Bello. "Measuring Structural Similarity in Music". *IEEE Transactions on Audio, Speech, and Language Processing*, Vol. 9, No. 7, pp. 2013–2025, 2011.

[3] AHRC Research - Centre for the History and Analysis of Recorded Music. "Mazurka project". url: www.mazurka.org.uk/ (accessed 24 May, 2016)

[4] J. J. Carabias-Orti, F. J. Rodriguez-Serrano, P. Vera-Candeas, N. Ruiz-Reyes, and F. J. Canadas-Quesada. "An audio to score alignment framework using spectral factorization and dynamic time warping". *International Society for Music Information Retrieval Conference*, pp. 742–748, 2015.

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

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

[7] P. Lamere. "The Infinite Jukebox", url: www.infinitejuke.com/ (accessed 24 May, 2016).

[8] B. McFee and D. P. W. Ellis. "Analyzing song structure with spectral clustering", *International Society for Music Information Retrieval Conference*, pp. 405–410, 2014.

[9] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and M. B. Westover. "Exact discovery of time series motifs", *SIAM International Conference on Data Mining*, pp. 473–484, 2009.

[10] A. Mueen, K. Viswanathan, C. K. Gupta and E. Keogh. "The fastest similarity search algorithm for time series subsequences under Euclidean distance", url: www.cs.unm.edu/~mueen/FastestSimilaritySearch.html (accessed 24 May, 2016).

[11] M. Müller. "Dynamic time warping". *Information retrieval for music and motion*, pp. 69-84, Springer, 2007.

[12] M. Müller, F. Kurth, and M. Clausen. "Audio matching via chroma-based statistical features". *International Society for Music Information Retrieval Conference*, pp. 288–295, 2005.

[13] 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.

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

[15] J. Serrà, 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.

[16] J. Serrà, X. Serra, and R. G. Andrzejak. "Cross recurrence quantification for cover song identification". *New Journal of Physics*, Vol. 11, No. 9, pp. 093017, 2009.

[17] 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". *International Society for Music Information Retrieval Conference*, pp. 95–10, 2014.

[18] D. F. Silva, V. M. A. Souza, and G. E. A. P. A. Batista. "Music shapelets for fast cover song recognition". *International Society for Music Information Retrieval Conference*, pp. 441–447, 2015.

[19] D. F. Silva, C.-C. M. Yeh, G. E. A. P. A. Batista, E. Keogh. "Supporting website for this work", url: http://sites.google.com/site/ismir2016simple/ (accessed 24 May, 2016).

[20] 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". *Journal of Information Science and Engineering*, vol. 24, No. 6, pp. 1669–1687, 2008.

[21] M. Wattenberg. "Arc diagrams: Visualizing structure in strings". *IEEE Symposium on Information Visualization*, pp 110–116, 2002.

[22] M. Wattenberg. "The Shape of Song", url: www.turbulence.org/Works/song/ (accessed 24 May, 2016).

[23] H. H. Wu, J. P. Bello. "Audio-based music visualization for music structure analysis". *Sound and Music Computing Conference*, pp. 1–6, 2010.