A CROSS-PLATFORM, STANDALONE APPLICATION FOR AUDIO RETRIEVAL BY RHYTHMIC AND TIMBRAL SIMILARITY

Aneesh Vartakavi, Cameron Summers

Georgia Tech. Center for Music Technology (GTCMT)

ABSTRACT

This paper details the implementation of a standalone desktop application in C++ for retrieval of audio files by rhythmic and timbral similarity, notable for being, to our knowledge, the first application of its type under the GNU General Public License (GPL). The implementation focuses on audio loops - short audio files intended to restart immediately upon completion for compositional purposes - to address the difficulty of navigating large loop libraries manually. The audio similarity measure utilizes tempo detection and statistical features of spectral frame periodicity to provide a ranking of most similar audio files to a particular reference file. Although similarity measures are somewhat subjective and difficult to evaluate, we propose two evaluations: 1) A classification accuracy on a ground truth data set developed by the authors using real-world audio loops and 2) A synthetic ground truth data set developed by the authors using related beat patterns at different tempos that demonstrates tempo invariance in retrieving rhythmically similar audio. (HOW ABOUT ONE LINE HERE TALKING ABOUT THE FINAL OUTCOME?)

Index Terms— audio similarity, audio retrieval, distance matrix, beat spectrum, tempo detection, C++, JUCE

1. INTRODUCTION

With advances in technology, large libraries of audio loops on a single hard drive have become commonplace. Navigating these libraries becomes increasingly difficult because knowledge about the files must be manually tagged or inspected by ear. A computational approach to audio retrieval via a similarity measure dramatically reduces the time spent navigating these libraries.

In this paper, we detail an implementation of a standalone desktop application in C++ for retrieval of audio loops by rhythmic and timbral similarity. A common task is finding an audio sample to sound simultaneously or in succession with a particular reference sample. The application allows a user to filter loops outside an acceptable range of tempo along with rhythmic and timbral distance given a particular reference loop. The tempo detection uses a simple approach based on characteristics of audio loops and the rhythmic distance is based on features derived from spectral periodicity. The timbral measure is based on euclidean distance between features derived from the Mel Frequency Cepstral Coeffecients (MFCC).

Our software provides the user with an interface to query for similar loops using a slider interface to specify a range of tempo, timbral and rhythmic distances. We believe this application has the potential to significantly reduce the amount of time spent searching a large database for similar loops.

2. RELATED WORK

Several studies exist on audio retrieval by rhythmic similarity using statistical features since this approach requires limited or no human input to operate. One successful instance uses correlated energy peaks across frequency sub-bands [1] while another uses a feature vector from a low frequency-weighted Short-Time Fourier Transform (STFT) [2]. Another approach using spectral periodicity called the Beat Spectrum [3] is utilized for rhythmic similarity in this application, because it is based on self-similarity of the audio, no particular features are required of the audio such as silence or periodic energy peaks - Only repetitive events are required, which are common in audio loops. We derive a periodicity measure from the similarity matrix similar to [3] for our application.

MFCC based timbre space has been found to be a good model for perceptual timber space [4]. MFCCs have since been used to create similarity measures [5] for genre recognition [6], [7]. We derived a simple timbral distance measure based on the euclidean distance between a feature vector derived from the MFCC's.

Several closed-source applications exist for determining audio similarity in libraries of audio files such as *Similarity* [8]. However, *Similarity* and most others are meant primarily for file library organization (e.g. removing duplicate files) and not for rapid navigation for compositional purposes. Another open-source command line application under a BSD-style license written in Python, AudioCompare [9], utilizes indices of dominant frequencies of the STFT for a general similarity measure, but this is not a musically descriptive measure and unreliable for our use case.

3. METHODOLOGY

In this section, we describe the implementation of our algorithm, particularly the tempo estimate, rhythmic similarity and the timbral similary measure. We also briefly discuss a GUI implementation to navigate a library using the aforementioned measures.

3.1. Tempo Estimation

WHAT DO YOU THINK ABOUT SEPARATING TEMPO ESTIMATION AND RHYTHMIC SIMILARITY?

We were going to say that the rhythmic similarity is tempo invariant, so I think we should separate the two. I'm fine with whatever you decide in the end.

3.2. Rhythmic Similarity

The rhythmic similarity recommendation algorithm is composed of two primary components that can be applied together or independently in the application: tempo estimation and distance between feature vectors of the spectral periodicity. For tempo estimation of an audio file, we use a simple method shown in equation (1) where T is the tempo, n is the number of expected beats in the loop, f_s is the sampling frequency of the audio, and N is the number of samples. This method is based on an assumption that the length of the audio loops in beats will always be a power of 2 of a 4/4 measure - a group of 4 beats - which is generally true for many loop libraries. Additionally, we scale the tempo to a range of 50-160 beats per minute.

$$T = 60 * n * f_s * 1/N \tag{1}$$

While there can be errors with this method when the assumptions do not hold, it worked consistently well for the real-world EarSketch [10] loops used for evaluation.

The second component of rhythmic similarity comes from a calculation of statistical features of spectral periodicity, called the Beat Spectrum by [3] who provides a detailed overview. We calculate the spectral periodicity as described in [3] and include some processing steps prior to computing features. The spectral periodicity of an audio file is derived from a similarity matrix S where S_{ij} is the cosine distance between the STFT of i-th and j-ith block of the audio file. A block size of 1024 and hop size of 512 were used for the STFT. At a typical sampling frequency of 44100Hz for the audio loops we tested, this provides resolution in time of 23ms, which is sufficient for detecting rhythms even at faster tempos that adequately describe the audio file. Variations on these values could be tested in future research to determine if optimization is possible.

Next each super diagonal of S is summed as in equation (2) to give a representation of the spectral periodicity in the lag domain, where l is lag and k is the block index. An example of the result is shown in Fig. 1, where the periodicity of events in the audio is evident. Alternatively, the spectral periodicity can be viewed as an autocorrelation of a vector of elements where each element is the STFT of a block of samples in the audio file.

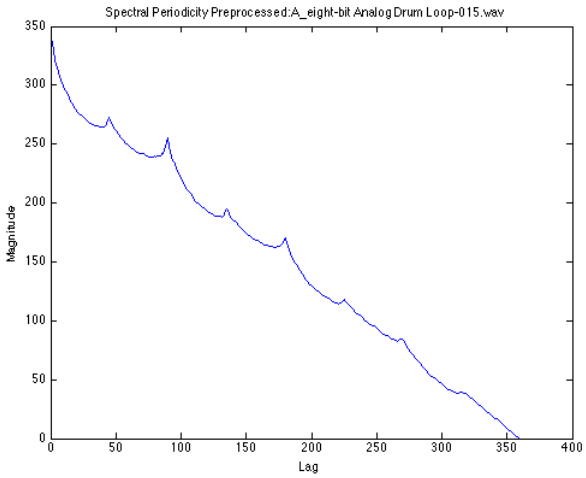

Fig. 1. Spectral periodicity measure for a sample loop

$$SP(l) = \sum S(k, k+l) \tag{2}$$

In the implementation in application, the spectral periodicity was calculated using a single similarity matrix with a length equal the number frames in the entire audio file. The audio file could alternatively be segmented and the spectral periodicity calculated on each segment as described by the Beat Spectrogram in [3], but since audio loops are often taken as a single unit for the compositional use case, our approach seems more appropriate for the application. Future implementations might explore this alternative and compare the performance for similarity ranking.

Prior to calculating features on spectral periodicity signal some preprocessing steps were taken to increase robustness of the measure. First, since the spectral periodicity is an autocorrelation and the number of elements summed decreases as the lag increases, a line was fit using least squares and subtracted from the signal. Next, the signal was normalized and inverted so that lags of lower distances had higher values by subtracting the signal from an equally long signal with every value a 1. Finally, a three point moving average filter was used for smoothing. Fig. 2 shows the post-processed signal.

After the preprocessing steps, a feature vector was constructed to describe the signal and these features were standardize by removing the mean and dividing by the standard deviation of each feature. A list of these features are shown in Table 1. Lastly, the algorithm determines a rhythmic distance measure by calculating the euclidean distance between

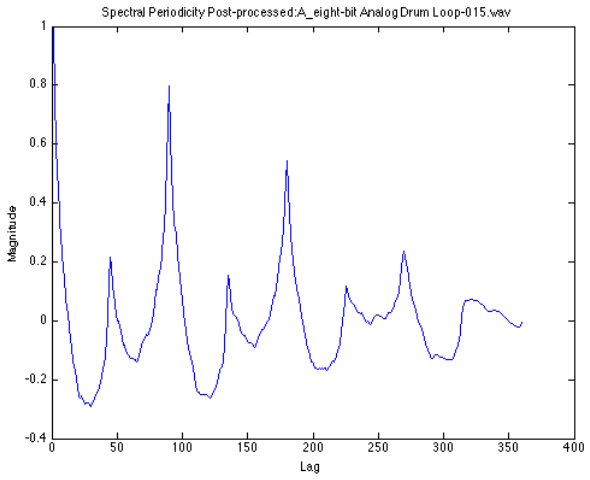

Fig. 2. Spectral periodicity measure after pre-processing

spectral periodicity feature vectors of two audio files.

Line Slope (LS)
LS y-intercept
Instaneous values at beats (assuming 4/4)
Standard Deviation
Skewness
Kurtosis
Sum of absolute value of difference between LS line and signal
Number of changes in signal slope

Table 1. Features Derived from Spectral Periodicity

3.3. Timbral Similarity

The timbral distance measure was based on thirteen dimensional MFCCs, which were computed as described in [5]. Additionally, we also computed the first derivative of the MFCC coefficients, storing a 52 dimensional feature vector of the mean and standard deviation of the MFCCs and their derivatives for each audio file. The timbral distance measure was formulated as the Euclidean distance between the aforementioned feature vector.

3.4. Implementation

Processing spectral periodicity features on large libraries of files requires speed and this informed the application design decisions. We chose the C++ framework JUCE [11] because C++ code can be optimized efficiently for time and memory and JUCE is highly regarded in the audio software development sphere for its user interface styling and audio utilities. Additionally, JUCE allows for multi-platform development and deployment, making greater the potential future dissemination of the application. We also use the Eigen Library[12], which provides optimized linear algebra routines and an FFT wrapper.

3.5. User Interface

Given the difficulty of navigating large loop libraries, the user interface of the application seeks to assist the algorithm in making this process simple and fast. A user can select a reference file in a list box on the left on which to compare other files in the library and has the option of listening to that file. Next there are three range sliders available; one for tempo, one for rhythmic distance and one for timbral distance. Then Similar files are automatically shown or removed in the right box according to the slider positions and those can also be played back by the user. A screenshot of the application is shown below in Fig. 3. The application also writes previously calculated features for a directory in a cache and allows the user to load and reuse that cache to avoid calculating features on the directory again. The cache, however, forces a re-computation when files are changed within the directory.

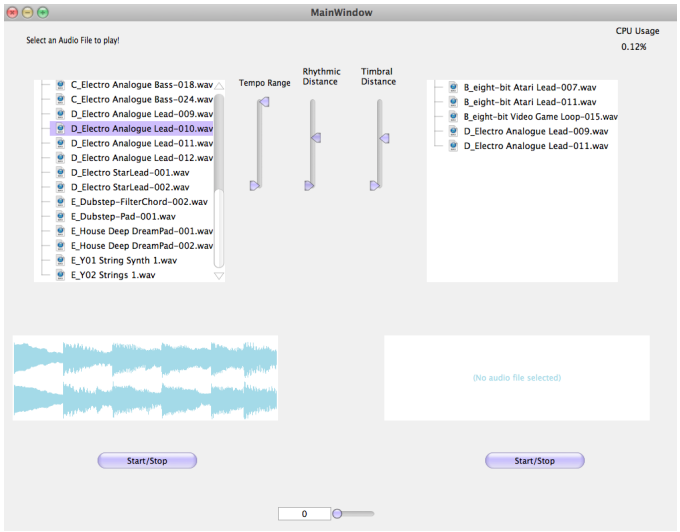

Fig. 3. Application GUI Screenshot

4. EVALUATION

We propose two methods for evaluation rhythmic similarity functions implemented in the application. Unable to find an accepted ground truth dataset for rhythmic similarity, the authors developed a ground truth set by selecting manually 30 loops from the EarSketch library and separating them by ear into 5 distinct rhythmic classes each with 6 loops. For each loop L, distances were calculated to the other 29 loops and ordered from the least distance to greatest. Then for each

of the smallest K distances an accuracy was recorded as the number of loops matching the class of L divided by K. Fig. 4 shows the ordered distances and associated classes for one audio loop in the ground truth seen in the figure as the loop with a distance of zero.

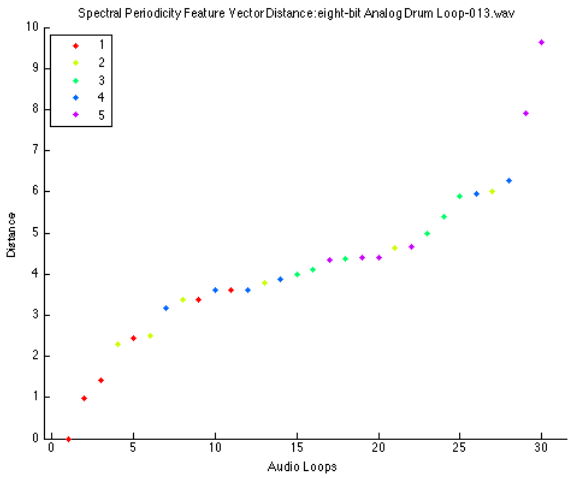

Fig. 4. INSERT CAPTION

Table 2 shows the average accuracy for all 30 loops for varying K. In other words, the top recommended loop was of the same class as the reference loop 80% of the time.

K=1	K=2	K=3	K=4	K=5
80%	70%	66%	57%	55%

Table 2. Average Class Accuracies

A second evaluation for tempo detection used the same 30 loops from the EarSketch library for which the loops had 7 unique tempos. The tempo detection accuracy for these 30 loops was 29 correct out of 30, or 97%, and the single incorrectly calculated tempo was off by 1 bpm.

5. DISCUSSION

The objective of providing a user a computational approach to and software application for navigating a loop library was fairly successful. As seen in Table 1 the most highly recommended loop was of the same classes as the reference loop 80% of the time. From the standpoint of the user, this significantly reduces the time spent looking for a similar loops that may complement one another musically. Care must be taken though, because although the authors have musical training, biases or inaccuracies may exist in the ground truth data. A more comprehensive future evaluation might include multiple ground truths from different musicians and loop libraries.

The tempo detection method works quite well in testing as indicated by the 97% accuracy in detecting the tempo on the ground truth. However, the assumption that any loop in a large library is a multiple of a power of 2 of a 4/4 measure may not always hold in real-world cases. The loop may be slightly longer or shorter, as is likely the case in the single incorrectly calculated tempo in the previous section, or in 3/4 time and small changes such as this make a large impact in selecting loops to form a composition. Since the user interface allows for a variable range in tempo, the user has significant flexibility in overcoming shortcomings such as this. Future implementations of tempo detection would likely benefit from a more robust system.

Future work could include several improvements to the existing algorithm and application. For example, additional features could be calculated and principal component analysis could be used to determine the most useful features. Additional similarity components could added such as chroma-based key detection since the knowledge of key would be quite useful for this use case. And other user interfaces such as self-organizing maps of the audio library may yield intuitive and creative ways for composers to select loops for a composition.

6. REFERENCES

[1] Eric D Scheirer, "Tempo and beat analysis of acoustic musical signals," *The Journal of the Acoustical Society of America*, vol. 103, pp. 588, 1998.

[2] Erling Wold, Thom Blum, Douglas Keislar, and James Wheaton, *Classification, search and retrieval of audio*, vol. 10, chapter, 1999.

[3] Jonathan Foote and Shingo Uchihashi, "The beat spectrum: A new approach to rhythm analysis.," in *ICME*, 2001.

[4] Hiroko Terasawa, Malcolm Slaney, and Jonathan Berger, "The thirteen colors of timbre," in *Applications of Signal Processing to Audio and Acoustics, 2005. IEEE Workshop on*. IEEE, 2005, pp. 323–326.

[5] Jesper Højvang Jensen, Mads Græsbøll Christensen, Daniel PW Ellis, and Søren Holdt Jensen, "Quantitative analysis of a common audio similarity measure," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 17, no. 4, pp. 693–703, 2009.

[6] Mark Levy and Mark Sandler, "Lightweight measures for timbral similarity of musical audio," in *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*. ACM, 2006, pp. 27–36.

[7] Jesper Højvang Jensen, Mads G Christensen, Manohar Murthi, and Søren Holdt Jensen, "Evaluation of mfcc estimation techniques for music similarity," 2006.

[8] GAR Software, "Similarity," http://www.similarityapp.com/, Dec. 2013.

[9] Zheng Hui Er An Dang, Cory Finger and Charles Connell, "Audiocompare," https://github.com/charlesconnell/AudioCompare, December 7 2013.

[10] Georgia Tech. Research Corporation, "Earsketch," http://earsketch.gatech.edu/, 2013.

[11] Raw Material Software Ltd, "Juce : Jules' utility class extensions," http://www.juce.com/, Dec. 2013.

[12] Gaël Guennebaud, Benoît Jacob, et al., "Eigen v3," http://eigen.tuxfamily.org, 2010.