Brian Siao Tick Chong

5/10/17

CS591 S1 Audio

Analyzing Changes in Music via Similarity Matrices and Novelty Scores

Options available to change in the code, “novelty_scores.ipynb”:

Part 1: Generating the self-similarity matrix M

- SR (sample rate of the song)
- W (size of window in samples for generating the chromas and self-similarity matrix)
- upperBound (for the range of frequencies to detect)
- startOctave (for the range of frequencies to detect)
- distanceMeasure (only cosine similarity is implemented but other distance/similarity metrics can be used)
- spectrum (when chromas are being formed in the function “fillChroma”, the spectrum can be squared, cubed, logged, or remain the same)
- length (the length of the file to analyze in seconds)
- musicdirectory (can specify here which file path the music is located in)
- title (file name of the music to analyze (must be in mono and a 16-bit wav file)
- start (start time in seconds of the segment of the file being analyzed)
- colormap (changes colormaps used to color the self-similarity matrix)

Part 2: Calculating Novelty Scores

- K_list (list of kernel sizes to calculate and graph novelty scores)
- gaussian (option to use a gaussian kernel)
- xaxis_scaler (smaller values generate less ticks on the x-axis, larger values generate more ticks on the x-axis)

Part 2.1: Animating Novelty Scores

- K_to_trace (specify which kernel size on the plot(s) generated in Part 2 you want to follow in real time!)
- Must install FFMPEG and add it to the PATH environment variable (for Windows)
- Videos will appear in the same directory as the code with the suffix: title + “_novelty-scores-animation.mp4” as well as play in the iPython notebook

Note: novelty_scores.ipynb requires the file cs591Utilities.py included in the .zip file.

The following report explains the thought process behind choices in some of the options with various examples, in addition to explanation of the theory behind the code. The theory already well-explained in three papers referenced at the end. In general, all options must be adjusted per song or genre of music.

All figures were generating using the code in “novelty_scores.ipynb” and are included as individual files with this report so that they can be viewed in full size.

Part 1: Generating the self-similarity matrix M

The current self-similarity matrix implementation relies on bins of semitones when generating the chromas that the cosine-similarity scores are based off of. The changes in verses can still be seen, as well as variations such as a vocal saying something, but because the current implementation relies on notes, most of the chromagraph/self-similarity matrix looks very similar for songs which consistently use a large spectrum (for example, drums). Figure 1 is an example from “Smoove – I’m a Man” seconds 3.0-23.0:

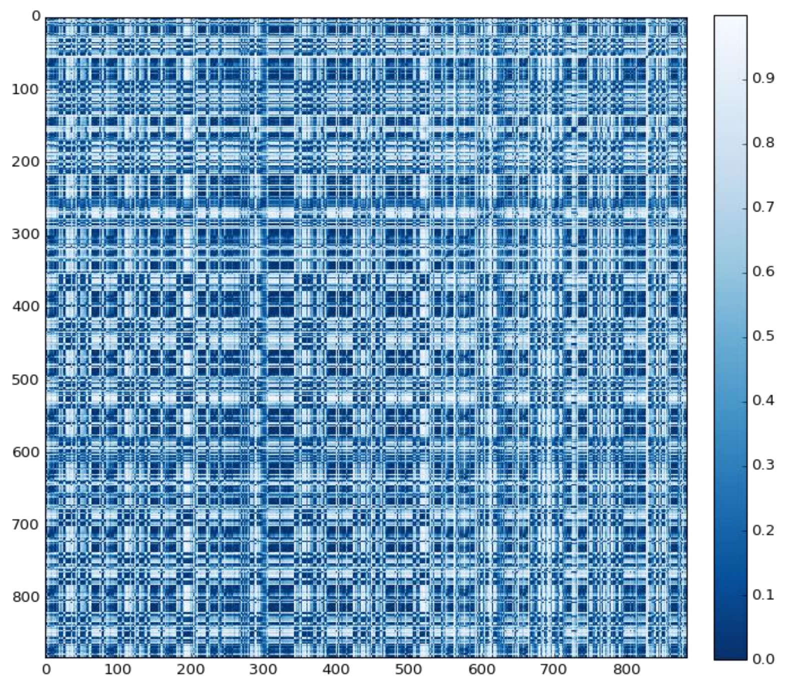

Figure 1 smoove-cube_window1000.png

This has a window size of 1000 with spectrum=cubed where blue = more similar, but since the bass guitar is present throughout the entire 20 seconds, the verses are difficult to distinguish. Here is the same clip except with spectrum=norm, window=2205, and cmap=”Spectral_r” where red is less similar and blue is more similar:

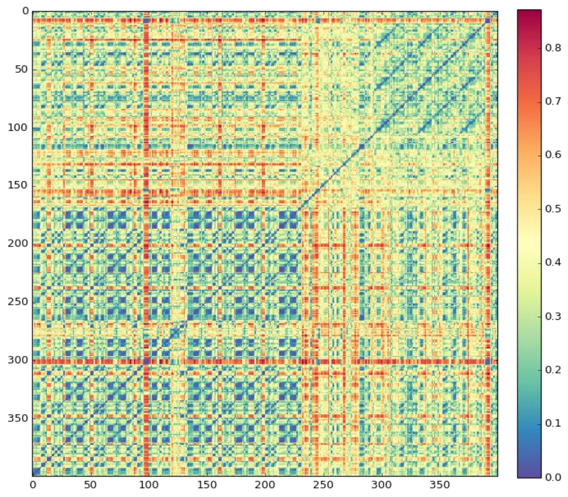

Figure 2 smoove-norm_window2205_3-23.png

This clearly shows when a voice appears causing the sizable section of yellow in the middle from ~500-620, and then drums and a shaker are added to the track at about window 520. Here is a graph of the same song except seconds 14.0-34.0:

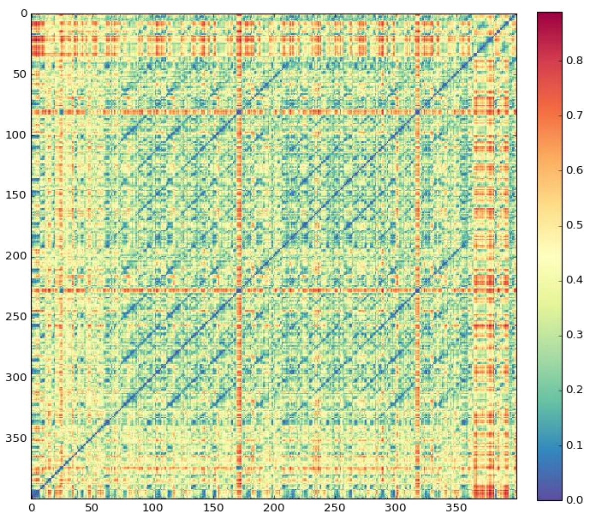

Figure 3 smoove-norm_window2205_14-34.png

The same part where the vocal is added, is at the very bottom left from roughly windows 0-60. The graph is yellow for the most part, with the repeated bass lines showing most prominently in blue, with two skinny orange strips representing whenever the bass plays a high note at those two exact moments (and the cross sections of these orange strips are dark blue). The bass then stops playing at window ~360 shown by the loss of blue. When comparing this figure with Figure 2, the blue spots are less defined, but nice diagonal lines appear.

So in summary, the distinctions between instrument additions and subtractions can be seen, but could use more clarification. The drums muddle the frequencies together because of its mix of frequencies, but the bass can still be seen due to its lower frequencies.

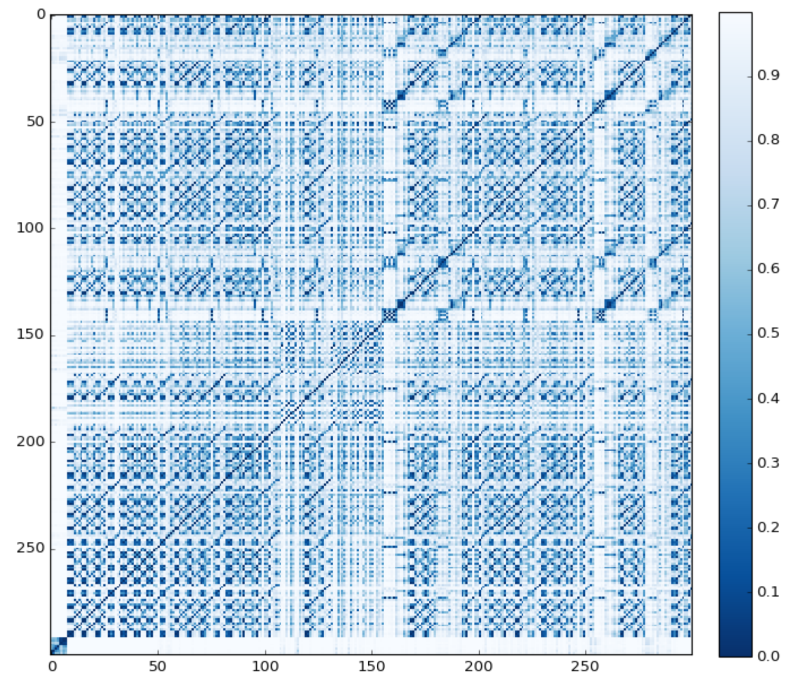

Figure 4 smoove-cube_window13230.png

Calculating and plotting novelty scores in the next section makes use of these similarity matrices. This helps determine musical section changes where one may be lost visually.

Part 2: Calculating Novelty Scores

In order to calculate novelty scores, the range of similarity values in the similarity matrix must be [-1,1], where -1 is the least similar and 1 is the most similar. The range of similarity is normalized to this range to remove dependence on the magnitude of the spectrum.[1] The method for calculating cosine similarity this way is described in the function "cosineSimilarity" in the code. With the range being [-1,1], we can then construct a unit kernel which has the format:

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

This kernel is then multiplied element-wise by a window of the similarity matrix sliding along the diagonal, and then the sum of those products forms the novelty score. The idea is that the identity line in the similarity matrix where the spectrum is compared to itself always lines up with the positive 1s, whereas the negative 1s will be multiplied by an area of the similarity matrix that represents the current spectrum being compared with a section of the music close-by. For example, if the entire window of the similarity matrix is extremely similar (similarity values of all positive ones), then multiplying that window element wise with the kernel results in a novelty score equal to 0:

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \text{ which sums to 0.}$$

On the other hand, if the window represents a changing spectrum, then the similarity matrix would have smaller or negative values perpendicular to the diagonal, resulting in a novelty score of > 0:

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \text{ which sums to 4.}$$

Essentially the novelty scores are exaggerated by the negative values being multiplied by each other in the case that the current window is relatively dissimilar.

These novelty scores can then be normalized to a range of [0:1] and the size of the kernel can be enlarged so that larger sliding windows can be analyzed. The purpose of a larger window is to analyze changes in music measures or even entire sections, whereas small windows will be sensitive to individual note or chord changes. Enlarging the kernel is simply performed by creating the Kronecker product of the unit kernel seen above with a matrix of ones (with the dimensions you want). For example:

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

Gaussian kernels seem to make the larger kernel sizes more sensitive to changes while still not being as sensitive as smaller kernels. Peaks are strengthened, while fewer peaks are added.

---

[1] https://infoscience.epfl.ch/record/190844/files/2013_Audio%20Novelty-based%20Segmentation%20of%20Music%20Concerts.pdf

So by sliding a window of dimensions of the same size as the kernel along the diagonal of the self-similarity matrix, novelty scores are generated throughout the song and then plotted. The following three examples are of self-similarity matrices and plots generated with novelty scores (functions “noveltyScores” and “plotNoveltyScores”). The options used to generate these examples are included in the title of each novelty score plot. It is recommended to listen to the songs (all on Youtube) and follow along the peaks yourself, and to open the full images included (otherwise it is impossible to see the time marks). **Part 2.1** of the code can generate video files which trace the plot in real-time which helps a lot for this. An example of a video file generated with this code is also included.

(Figure 5 and Figure 6) For “You and I” by Jacob Collier, the kernel size of 16 (orange in Figure 6) showed sensitivity/peaks whenever there were chord changes, whereas the kernel size of 64 (blue in Figure 6) mainly showed sensitivity whenever there were major key changes.

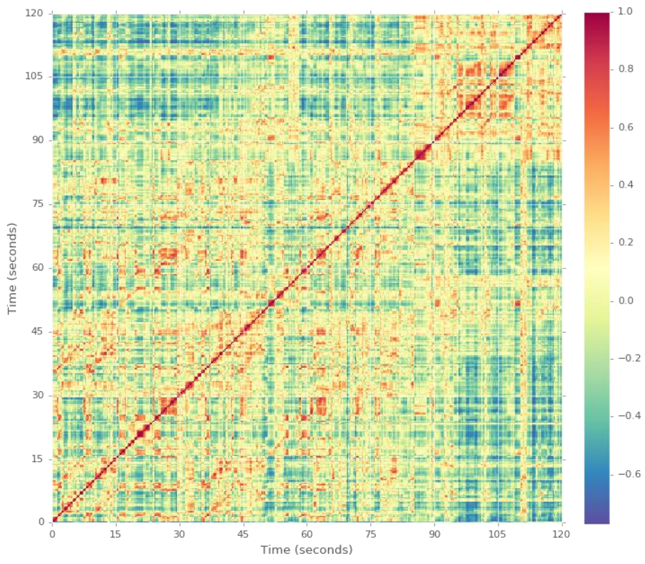

Figure 5 youandijacob-norm-matrix.png

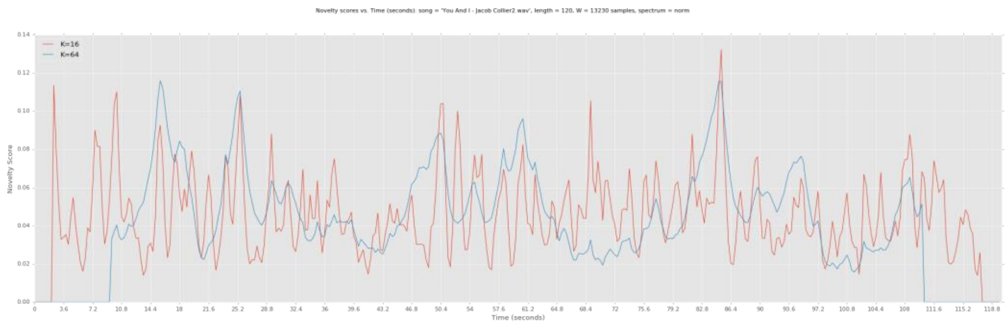

Figure 6 youandijacob-norm-novscores.png

(Figure 7 and Figure 8) The novelty scores for “Phunkdified” as performed live by Ben Lapps shows clear changes in music sections due to the repetitive nature of each section. The changes are most easily seen with a window size of 13230 samples, squared spectra, and a kernel size of 32 (blue in Figure 8):

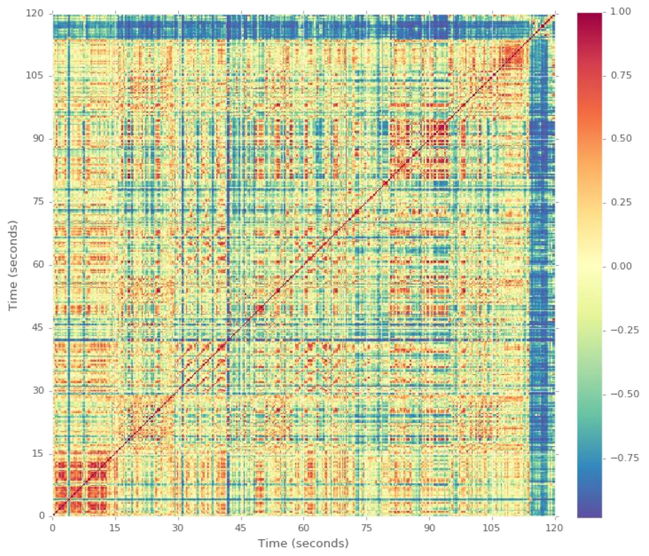

Figure 7 phunkdified-squared-matrix.png

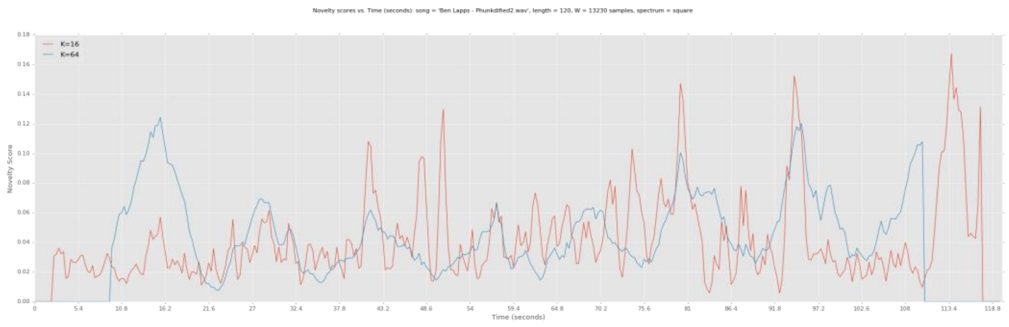

Figure 8 phunkdified-squared-novscores.png

(Figure 9, Figure 10, Figure 11, Figure 12) Jacob Collier’s cover of “Don’t You Worry ‘Bout a Thing” was especially interesting because of the large number of key shifts and musical sections. Due to the song being 6 minutes long, a larger window size was used to generate the similarity matrix. As a result, a kernel of size 64 (blue) was too large to accurately judge the section changes in the song, and instead, a kernel of size 32 (purple) found the changes the best. The kernel size of 16 (orange) was also better than the size 64 kernel, but was too sensitive. Interestingly, many of the peaks shown for the size 32 kernel are double peaks, indicative of each music section change being preempted by a harsh transition.

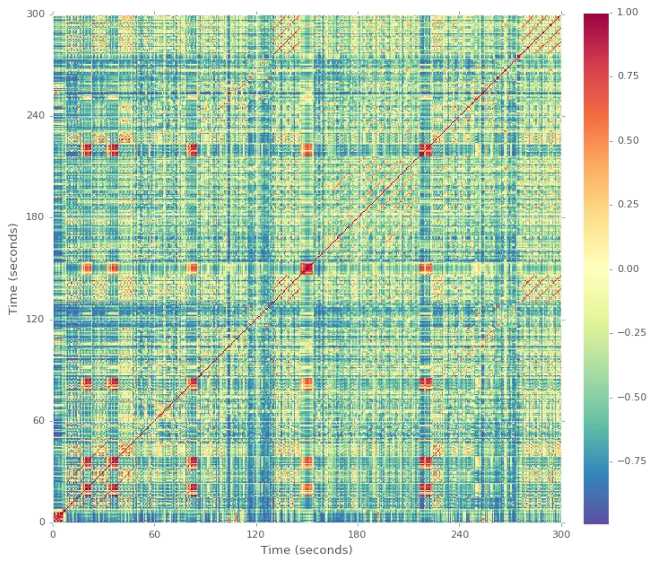

Figure 9 dontuworryjacob-squared-matrix.png

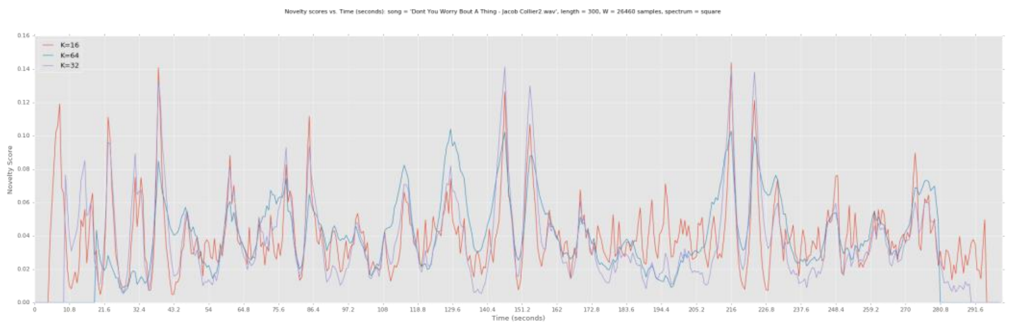

Figure 10 dontuworryjacob-squared-novscores.png

Unlike other songs, the musical sections for this song can be more easily seen when not squaring the values of the spectrum when forming the similarity matrix, resulting in a more similar matrix overall:

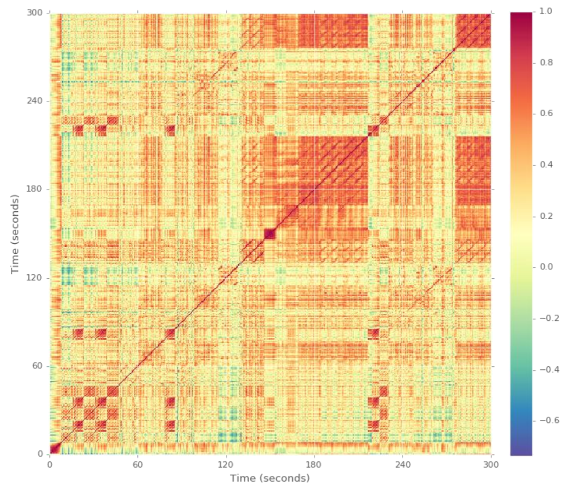

Figure 11 dontuworryjacob-norm-matrix.png

Generating and using a more similar similarity matrix helps a song like this, which has such complicated and frequent changes, have less variable novelty scores when compared to Figure 10 dontuworryjacob-squared-novscores.png. This is better for finding the more drastic changes in the song, such as with novelty scores generated with the kernel of size of 48 (blue):

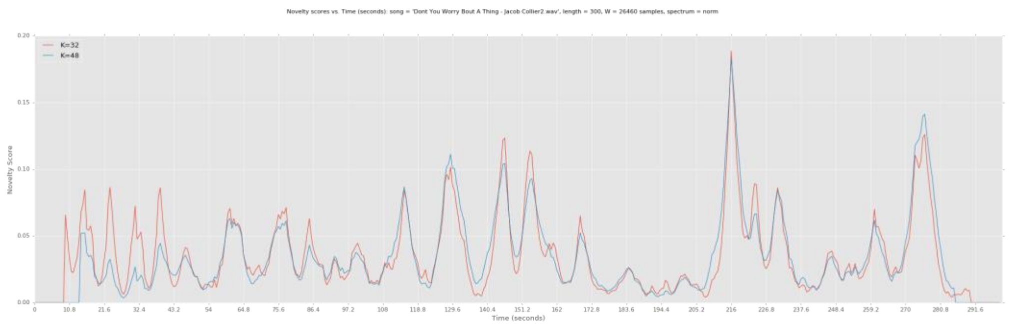

Figure 12 dontuworryjacob-norm-novscores.png

This kernel size 32 (orange) is much less noisy in Figure 12, making it easier to perform peak detection. Again, double peaks can be seen whenever a musical transition is preempted. The size 48 kernel (blue) has smaller peaks near the beginning where there are smaller changes from 10.8-43.2 seconds.

So ideally, the self-similarity matrix should have a decent amount of yellow if you do not want too much sensitivity to small changes. Additionally, the kernel can be too big or too small.[2]

[2] This paper shows how the ideal size of the kernel can be calculated: https://quod.lib.umich.edu/cgi/p/pod/dod-idx/music-structural-analysis-via-novelty-shape-detection.pdf?c=icmc;idno=bbp2372.2004.132

References

- http://musicweb.ucsd.edu/~sdubnov/CATbox/Reader/p77-foote.pdf
- https://www.fxpal.com/publications/automatic-audio-segmentation-using-a-measure-of-audio-novelty.pdf
- https://pdfs.semanticscholar.org/85f5/66664bc945f83dca776a322dc337900a8a86.pdf
- https://infoscience.epfl.ch/record/190844/files/2013_Audio%20Novelty-based%20Segmentation%20of%20Music%20Concerts.pdf
- https://quod.lib.umich.edu/cgi/p/pod/dod-idx/music-structural-analysis-via-novelty-shape-detection.pdf?c=icmc;idno=bbp2372.2004.132