The S-Matrix: A Novel Approach to Musical Segment Detection

Robert Turetsky
rjt72@columbia.edu

Abstract: This paper describes a method to detect segments within a piece of music. These segments are the characters and words that make up music, and they are found by analyzing the self-similarity of a waveform. The segments can be on any level, so this method is a reliable way of extracting notes or phrases from a musical signal. Some potential applications for audio segmenting, such as smart cut-and-paste and audio gisting, are also discussed.

I. Introduction

In text, it is possible to build words out of letters, sentences out of words, and paragraphs out of sentences. In video, a shot is constructed from frames, a scene is divided into carefully ordered shots, and an entire film is built up from different scenes. In music, we can say the same thing: A symphony is built up of movements, which are constructed from phrases, which in turn are built up from a sequence of notes.

In text, we can define word boundaries by using space characters. For example,

Digital Audio Processing

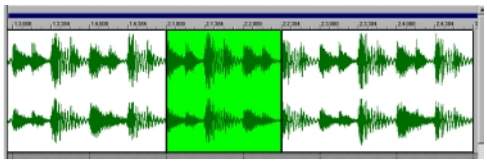
it is easy to pick out the word Audio in the above figure. This can be done by inspection or automatically, as the smart cut-and-paste function in Microsoft Word does.

In video, we can find segment boundaries by comparing the difference between one frame and the next.

For example, these adjacent shots from Stanley Kubrick s Eyes Wide Shut are a perfect example of what is called a reaction shot :


Here we see an action — a man walking with two women - and a reaction — his wife looking back at him in anger. Film audiences almost instinctively extract information from these juxtapositions of shots. While automatic scene extraction is still a current research topic (Kender and Yeo 1998), at least the language is well defined.

In music, we have no such luxury. Ideally, we would like to have something like the mockup in the following image:



The idea would be to have a smart cut-and-paste function, which would know when a musical phrase begins and ends. Thus, an artist would be able to manipulate a sound waveform at the segment level.

This would allow us to do some interesting thing with any given piece of music. For instance, it would be possible to provide a seek button for an audio browser to skip to the next original segment. Or this could be a first step in the blue-sky goal of automatic musical analysis, whose eventual goal is a music theory and artificial intelligence-based digital music composer. On a lower level, we would be able to automatically find the boundaries of individual notes in a song, so that an enterprising artist can sample it and come up with structures built from these samples.

At first glance, it may seem that silence detection will yield section breaks. This, in fact, is the approach that Sound Forge uses. However, silence detection falters in the face of noise or a piece without silence to signal the end of a segment.

By simply looking at a waveform, it is nearly impossible to tell where one phrase ends and another begins. Straightforward approaches like measuring spectral differences are not useful because they typically give too many false alarms as typical spectra for speech and music is in constant flux. (Foote, 1999)

Instead, we turn to the definition of a segment. The cornerstone of this system is the premise that there must be some feature — any feature — which most points in a segment share with each other.

Assuming that we have discovered that feature, we can make the following inference: Points in the same segment will have similar feature values. Points in different segments will have different features. Thus if we measure the similarity between one point and another, we can tell if they are part of the same section. The method to do so was described in Foote 1999.

The remainder of the paper is organized in the following manner: I will now describe the technique used to segment audio into phrases. I will then present some of the results I ve found. Following that, I will address some of the limitations of the system and some attempts at improving their performance. Finally, I will discuss some possible ideas that are built from the system I have constructed. Attached will also be a code listing and instructions to run it.

II. Methods

We encapsulate the differences between points in a musical signal in a structure known as the S-Matrix. We can then extract the novelty of a particular point from the S-Matrix.

Any musical signal travels through the following path on its way to segmentation:

1. Feature Extraction
2. Distance Measurement
3. Novelty Score Calculation
4. Peak Finding
5. Segment Boundary Finding
6. Application-Specific Post-Processing

I have built a system based on this path, I will now discuss each point on it.

1. Feature Extraction

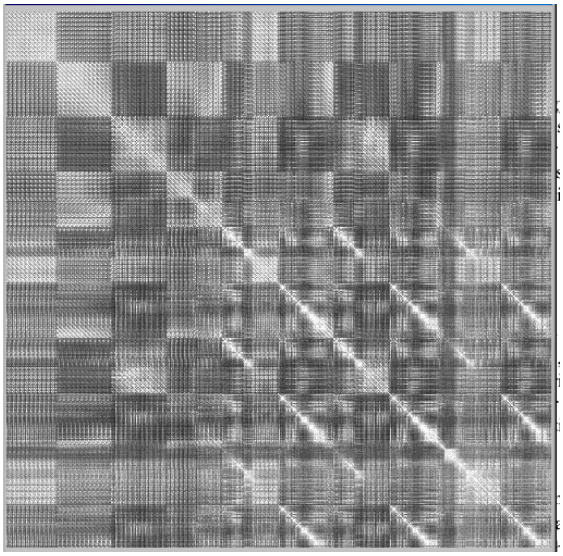
If we break up a musical signal into `maxt` chunks, we can construct a 2-D matrix of differences between features. This matrix is known as the S-Matrix (S for similarity). For example:

```
S[i, j] = diff(feature(i),
          feature(j))
```

This equation leaves a lot open to interpretation. What features should be used? How do we calculate the difference between features? What is an appropriate size of the matrix?

These questions can be answered empirically, based on the task at hand.

The S-matrix for the first 1:30 of Dream Your Dreams by Space Girl is below:



Dream2min.wav

In this graph, the timescale moves down and to the right. Similar points are white, and different points are black.

The first thing to notice about this structure is the white diagonal line going from the upper left corner to the lower right. This exists because every point is maximally similar to itself. The next thing to notice is the checkerboard pattern that is most clearly seen in the upper-left of the diagram. These white squares along the diagonal correspond to self-similar points of the same section. The dark squares show mismatches between points, saying that these points belong to different sections. Finally, notice the off-center diagonal lines. These correspond to phrases that are repeated.

To get the above S-Matrix, I divided the musical signal into 500 pieces and took the FFT. The system allows you to choose if you want to stagger large windows or use sequential smaller windows. Also, you have the option of using a Hamming window to avoid edge effects.

2. Distance Measurement

To calculate the difference between FFT points, I used the cosine difference. This is defined as:

$$D(i,j) = \frac{v_i ? v_j}{\|v_i\| \|v_j\|}$$

This distance is good because it makes large differences have small cosine differences, even if large values occur in the feature. The inverse is true for small differences.

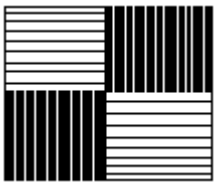
Now that we have the distances between points embedded in the S-Matrix, what do we do with it?

3. Novelty Score Calculation

Let s consider a piece that consists only of two different notes. The S-Matrix of that graph would look as follows:

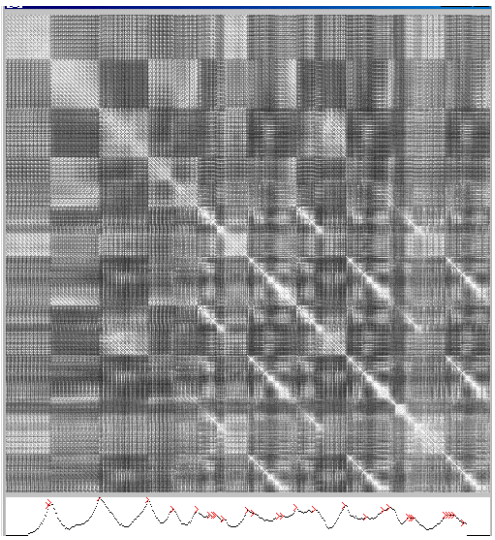
The upper-left and lower-right corners are white because each note is similar to itself. The upper-right and lower-left corners are black because each note is different then the other.

Now if we recall the theory that features of points in the same segment have (about) the same values, we can see that a signal with two different segments would looks something more like this:



Thus, the way to identify sections is to correlate the diagonals of the S-matrix with a checkerboard matrix.

We run the checkerboard matrix down the diagonal of the S-matrix, and the value of the correlation at each point is called the novelty score . The novelty score is a measure of how new a particular point is.



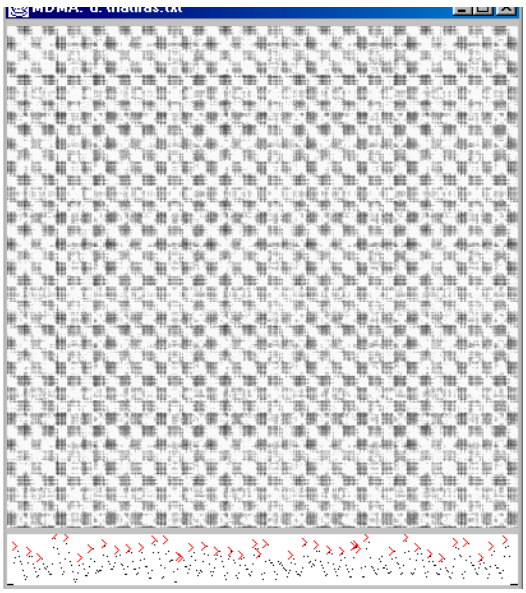
Underneath the S-Matrix of the Space Girl clip, you can see the novelty score.

Notice that the peaks correspond to points in the S-Matrix (along the diagonal) that look like the black and white checkerboard. These peaks are the segment boundaries.

dream2mSEG.wav

We can change the level of detail in the segments by changing the size of the checkerboard. For example, the above figure used a checkerboard size of 3 seconds, and it captured measure-sized phrases.

If we use a smaller checkerboard, we can capture even finer detail. For example, in this clip, Space Invaders by Hatiras, I have used a .25 second checkerboard to extract individual notes.



Here is the original 30 second clip:

Hatiras30.wav

Now here are some samples of individual notes taken from the clip:

hat1.wav hat2.wav hat3.wav

hat4.wav hat5.wav

Notice that clips 1 and 2 are of the same sound, though clip 2 has the added cymbal crash. The final two clips are both longer then one note. We can rectify the situation by using an even smaller checkerboard. The following three clips were taken with a checkerboard size of .1 sec. Notice that the individual notes of clips 4 and 5 have been isolated.

hat1down.wav hat2down.wav hat3down.wav

4. Peak finding

Once we find the peaks in the novelty index, we can divide a piece of music into segments.

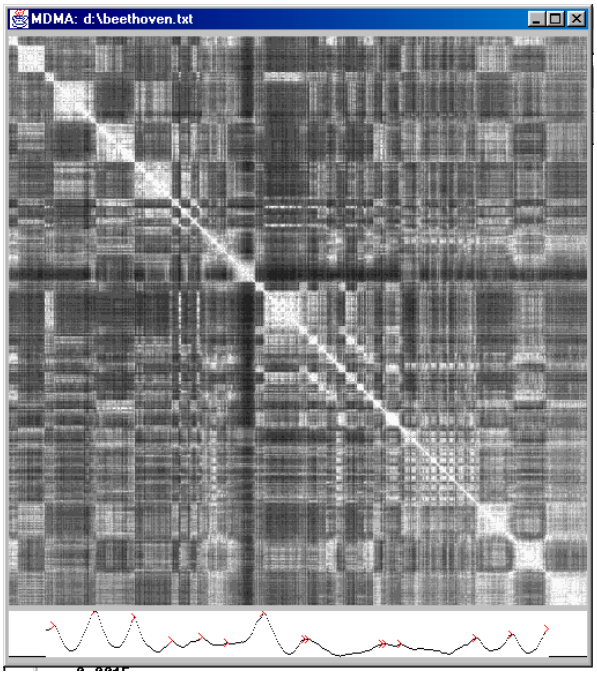
The trouble with doing simple thresholding is that the data is not very smooth and there are many false alarms. We cannot use an averaging filter because it will skew the timing of the peaks.

The method I took was to find points in the novelty score that are higher then their neighbors, are above global and local thresholds and are a certain number of points away from the last peak.

This, unfortunately, is still not enough. In fact, most of the errors in the above graphs can be traced to inaccuracies in the peak finding.

Just as an aside, this is something that humans manage to do with relative ease, however I have found it to be extremely difficult to pick peaks in a computer program.

For example, the following S-matrix is from Beethoven s 5th Symphony:



Beethoven.wav

Notice that all of the errors seem to come from peaks which should not have been chosen.

5. Segment finding

Once we have the peaks, finding the segments is simply a matter of placing the successive segments onto a queue and playing them out.

6. Application-Specific post processing

Once we have the music broken up into segments, we can do all sorts of things. For example, we can identify the segment which best characterizes the work (audio gisting) by finding out the segments which are most repeated.

Audio gisting would be useful for on-line music retailers, such as cdnow.com, who provide short clips of the tracks they sell. These clips, however, seem to be taken at random. With audio gisting, they would be able to automatically tune their clips to be the most descriptive of their work. The same applies to record stores that sell electronic dance music — with an enormous amount of vinyl being released every week, it is impossible for an enterprising DJ to listen to it all and still have time to eat and sleep. Thus, the DJ relies on word of mouth, getting white-labeled promos , and the fame of the musician. This means that many new artists and styles can get overlooked, simply because they are one of ten thousand new albums which have been pressed that week. If a store instead set up kiosks with sound clips of albums searchable by artist, genre, label, date released and number of remixes, the DJ will be able to browse through many more albums in one visit.

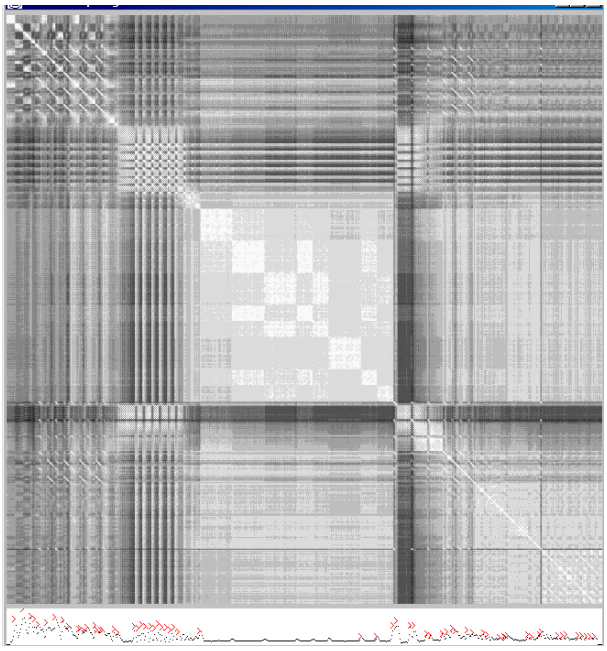
Another application may be for audio search and retrieval. This is because the S-matrix stores only the relative structure of a song, not the specific details. Thus, it is possible to imagine searching for a performance based on the S-Matrix of a MIDI file. This seems a little bit clunky, but it is an option.

III. Results

The only metric I can think of to compare my results would be comparing the software against human listeners. Unfortunately, I did not have access to the funding or the time to set up a study like that, so I had to rely on my own instinct.

I believe that most of my results came out matching my idea of where the segment lines could have been drawn. Most of my errors came from the peak picker. The worst error that I saw in the trial runs was when Space Girl was singing the words Dream your dreams . A human listener would have grouped the entire phrase together based on gestalt principles, however the computer saw the variation in the speech spectrum to be too great to consider part of the same segment.

In one example, however, the system seemed to fail. This was when we looked at the entire Space Girl song.

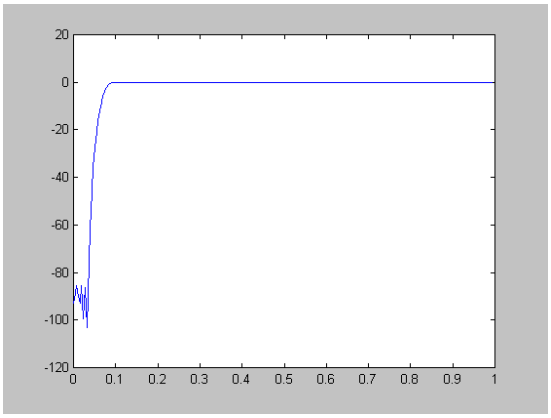


The upper-left hand corner is recognizable from the first Space Girl trial run. The lower-right hand corner corresponds to the dramatic conclusion of the piece. While both the beginning and the end are segmented properly, the big white square

in the center is where the system breaks down.

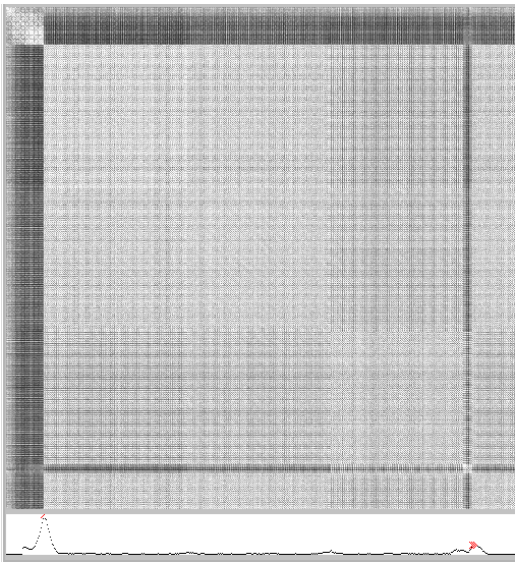
Notice that there are no segments found during the middle portion. I believe that this is because the bass drum seems to overpower the lower portion of the frequency spectrum.

The first thing I did to try and alleviate the problem is to filter out the bands where the bass drum is prominent. I designed a ~200 order linear phase filter, whose frequency response is shown here:



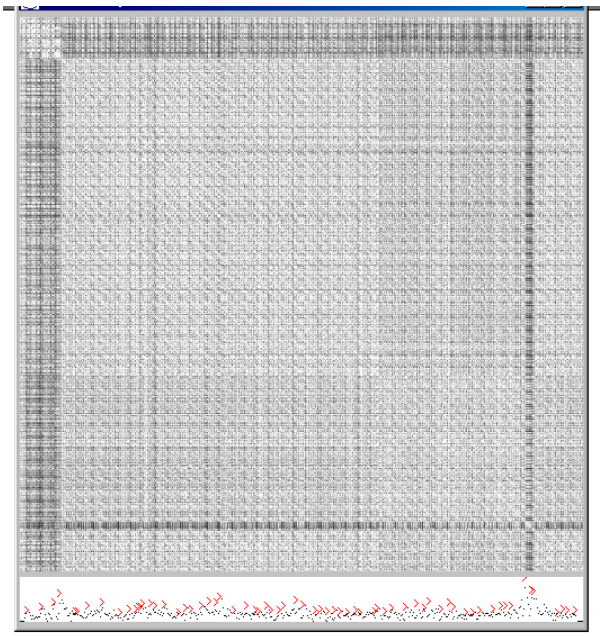
This, unfortunately, did little more than blur the S-matrix, and the whole thing still looked like one section.

The next thing I did was to try and increase the resolution in the S-matrix by focusing only on the middle section of the piece.



This also did little to alleviate the problem.

The final thing I tried to do was to switch the feature used. I posited that maybe the pitches of the segments were so similar that the distinction between pitches was lost in the FFT. I thus opted to use the Constant Q transform (Brown 1991), a logarithmic scale transform which maintains quarter-tone resolution.



This attempt also seemed futile, but for the exact opposite reason as the filtered version. In this case, the resolution was so good that there appeared to be too many little segments as opposed to one big one.

In the end, I went and listened to the clip a few times and decided that maybe it is all one big section

IV. Conclusion and Future Work

The S-Matrix structure is useful for automatically finding musical phrases in a number of situations. It can adapt well over a number of instrumental genres, such as classical, electronic, jazz and Muzak. It is not, as of yet, equipped to deal with

vocals because it does not perceptually group the same phrases together.

Finding musical phrases can be useful in its own right, or it can be used as a first step in more ethereal work such as CASA (Computation Auditory Scene Analysis). It may be easier to identify sound sources if we know where each sound source begins and ends.

The S-matrix does not explain human perception of music, however. This is for two reasons. First, it is not real-time, while our appreciation of music is. Second, the system does not group sounds based on gestalt principles. The effects of the latter can be seen in the first segmentation of Dream Your Dreams , as the voice was broken up into many different segments.

There are a number of improvements I would like to make on this work, ideas I would like to experiment with, and directions I would like to take it.

My immediate goal is to improve the peak-picking feature. This has proven to be the only thing keeping the system from living up to its potential.

The next thing I would like to do is to investigate alternatives to the novelty index. For example, finding long non-center diagonals of similarity will lead us to finding repeated segments. I would also like to investigate using Hopfield networks in order to detect similarity across musical inversions. Hopfield nets are self-associative networks, who have the property that they will settle into the training vector which most closely resembles their input. Thus, an inversion can be treated as some kind of musical noise to be filtered out.

Another interesting thing worth investigating is building a segment tree . Each musical phrase would have pointers to the lower-level phrases which it is built from. Thus it would be possible to traverse the tree from the song level all the way down to the note level.

Finally, for the more immediate goal of funding my education, I would like to implement the audio gisting concept on top of the S-matrix structure.

Bibliography:

Brown, Judith Calculation of a Constant Q Spectral Transform , Journal of the Acoustical Society of America pp. 425 - 435 1991

Foot, Jonathan Methods for the Automatic Analysis of Music and Audio , Xerox Park Technical Report 1999

Kender, J.R., and Yeo, B.L., "Video Scene Segmentation Via Continuous Video Coherence", CVPR ’98 (oral presentation) http://www.cs.columbia.edu/~jrk/research/video-coherence.ps

Code Listing

The following files are attached to this document, corresponding to the major classes being used by the system:
DisplayFrame.java — Used to set up the visualization of the S-Matrix, and as a Main class to kick off the S-Matrix process
SMatrix.java — This builds and stores the S-Matrix from a Signal. This class handles the cosine difference, calculation of the novelty index and outputting the section breaks as Matlab code.
Signal.java — Reads and stores a musical waveform (file formatting to be discussed below). This is responsible for feature calculation and encapsulating signal information (such as sampling rate, length, etc) for use in the SMatrix class.
Complex.java — a small complex number library
Time.java and **Section.java** — Data structures for manipulating time and sample numbers.

I have also attached two short Matlab routines:
wavprepare.m — Outputs a text file containing good waveform input for the system
hpf.m — Designs the high pass filter used in the Results section above.

All of these files were compiled and run on Java 2 (aka Java 1.2). They should be forwards compatible with Java 1.3, however DisplayFrame.java makes use of Swing, which is not compatible with Java 1.1.

The file format for a signal is a text file containing the Matlab variable which is the result of the wavread command. Thus, the text file is a list of real numbers between —1 and 1 which, if imported, will play the music file to be run through the system.

To run the program with a Java 2 (aka Java 1.2) Virtual Machine, type:

```
javac –classpath . DisplayFrame.java
java –classpath . DisplayFrame <filename> <fs> <tmax> <kernel>
```

where the command line arguments are:
<filename> - The text file containing the musical signal
<fs> - the sampling rate of the signal in Hz (ie 44100, 22050, 16000, etc .)
<tmax> - the size of the S-Matrix
<kernel> - the length (in seconds) of the checkerboard function to be used (ie 3.0, .25)

All other parameters must be altered from within the code, and thus the system must be recompiled in order for these changes to take effect.