

# Using Long-Term Structure to Retrieve Music: Representation and Matching.

Jean-Julien Aucouturier

Department of Electrical Engineering, King's College,  
London. Strand, London WC2 2LR. UK

Phone: + 44 20 7848 2041

[Jean-julien.aucouturier@kcl.ac.uk](mailto:Jean-julien.aucouturier@kcl.ac.uk)

Mark Sandler

Department of Electrical Engineering, King's College,  
London. Strand, London WC2 2LR. UK

Phone: + 44 20 7848 2041

[Mark.sandler@kcl.ac.uk](mailto:Mark.sandler@kcl.ac.uk)

## ABSTRACT

We present a measure of the similarity of the long-term structure of musical pieces. The system deals with raw polyphonic data. Through unsupervised learning, we generate an abstract representation of music - the "texture score". This "texture score" can be matched to other similar scores using a *generalized edit distance*, in order to assess structural similarity. We notably apply this algorithm to the retrieval of different interpretations of the same song within a music database.

## 1. MOTIVATION

Motivation for this system is our belief that a bird-eye-view of a song's long-term structure is often a sufficient description for music retrieval purposes. In particular, our system doesn't use any "transcription" information such as pitch or rhythm. Thus, it can deal with polyphonic music without the problem of instrument separation.

A similar approach has already been illustrated by Foote in [1], where the author designs an algorithm to retrieve orchestral music based on the energy profiles. A drawback of his system is that it requires music with high dynamic variations. To address this problem, our approach is rather based on spectral variation: we uncover and match the succession over time of abstract "spectral textures".

## 2. REPRESENTATION

A piece of polyphonic music can be viewed as the superposition of different instruments playing together, each with its own timbre. We call "texture" the polyphonic timbre resulting of this superposition. For example, a piece of rock music could be the succession over time of the following textures: {drums}, then {drums + bass + guitar}, then {drums + bass}, then {drums + bass + guitar + voice}, etc...

The front-end for our system is based on work done by the authors in [2]. The musical signal is first windowed into short 30ms overlapping frames. For each of the frames, we compute the short-time spectrum. We then estimate its spectral envelope using Mel Cepstrum Coefficients [3]. A Hidden Markov Model (HMM) [4] is then used to classify the frames in an unsupervised way: it learns the different textures occurring in the song in terms of mixtures of Gaussian distributions over the space of spectral envelopes. The learning is done with the classic Baum-Welsh algorithm. Each state of the HMM accounts

for one texture. Through Viterbi decoding, we finally label each frame with its corresponding texture.

Our "texture score" representation is just the succession over time of the textures learned by the model (figure 1). It reveals much of the structure of the song: phrases succeed to phrases, common patterns are repeated every verse and chorus, instrument solos stand out clearly and echo the introduction and ending, etc.

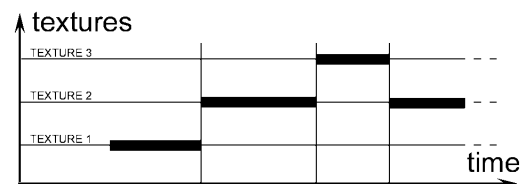


Figure 1: The texture score representation for a few seconds of music.

One interesting property of this representation is that it is based on spectrum, but is independent of what the spectrums really are: We only look at the succession of the textures, not at the textures themselves. The texture score of figure 1 could correspond to {drums} - {guitar + drums} - {guitar + drums + voice} - {guitar + drums}, but could also well be {cello} - {cello + violin} - {cello + violin + voice} - {cello + violin}, etc. We will use this property to match different interpretations of the same song (i.e. same long-term structure) which use different instrumentations (i.e. the spectral content of the textures is different).

## 3. MATCHING

In order to assess the structural similarity of pieces of music, we've designed an appropriate string-matching algorithm to compare texture scores. Each score is a simple string of digits out of a small alphabet: if we've identified 4 textures in the song, the score will be of the form ...11221333441... out of the alphabet {1,2,3,4}. There are three issues that the string-matching algorithm needs to solve:

- *Noise*: similar structures can differ quite a lot locally, so the matching can only be approximate.
- *Time Warping*: two different performances with the same structure can have a different rhythm or expressivity (rubato...).

- *Permutations*: the numeration of the textures by the front-end is arbitrary. This means that a texture which is referred to as “1” in one song, could be referred to as “3” in another. Therefore, the two strings “112133” and “331322” should be considered to be the same (as they differ only by the following permutation  $\{(1,3), (2,1), (3,2)\}$ ).

The first two issues are classically dealt with using dynamic programming to compute an edit distance (also called *Levenshtein distance*) [5]. It gives the minimal number of local transformations (insertion, deletion, substitution) needed to transform – or “edit”- one string into one other.

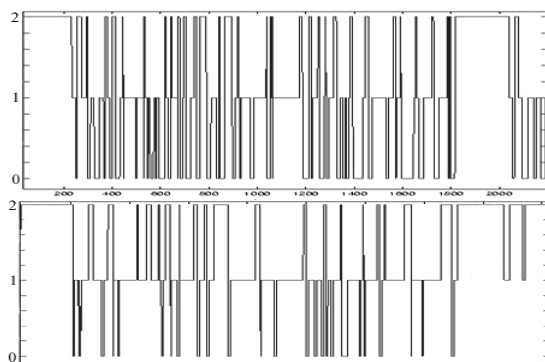
However, the third issue has not received much coverage in the string matching literature. To avoid the brute force approach consisting of  $n!$  distance measures for all permutation of the alphabet, Baker in [6] suggests an interesting factorization method. Unfortunately, it is mainly designed for exact matching (without noise), and is also very dependent on the time scale.

Our integrated solution to these three issues is a *generalized edit distance*, where we progressively adapt the cost of the each elementary substitution as the edit distance between two strings is computed. At the beginning of the process, we “charge” every substitution of one symbol into another, except the identity. By the end of the measure, the costs have changed to “learn” the best permutation between the two strings: we “charge” every substitution (including identity) except the ones corresponding to the permutation between the two strings.

## 4. TWO APPLICATIONS

### 4.1 Clustering covers of the same songs

Figure 2 shows the texture scores for the beginning of two versions of the same song, with different instrumentation: the first one is a male singer and an accompaniment based on accordion; the second one has a female singer and violins. Since we’ve freed ourselves from these spectral differences by using the texture scores, we are able to notice that the two pieces show some similarity. We have applied our algorithm on a database containing different versions of different songs, and the results are encouraging: the edit distance between “covers” is generally small, and the distance between different songs is big, which allows us to cluster the different interpretations.



**Figure2: Comparison of the texture score representations of two different interpretations of the same song.**

### 4.2 Clustering songs of the same genre.

We have also applied our algorithm to cluster a database containing acoustic blues, folk and pop pieces. As most of the blues tunes show a common phrase structure (AAB), we are once again able to gather and separate most of them from the other pieces. Once again, a bottom-up spectral approach can’t succeed in this task, since all the pieces contain mostly the same instrumentation (voice + guitar).

## 5. CONCLUSION

The texture score is a good representation to study the long-term structure of polyphonic musical signals. In the context of string matching, it provides an efficient retrieval tool to cluster songs with the same structure. Two applications are covers of the same tune, and pieces of the same “structural” genre.

This tool is especially useful since it disregards the spectrum content of the signals. Obtaining the same assessment of structural similarity from the extraction of “transcription” features such as pitch, instrumentation and rhythm would actually require very sophisticated high-level knowledge.

The generation of the texture score involves a machine-learning algorithm, which is quite intensive for a database application (processing a piece of music takes about real time), but once extracted, the score can be stored as metadata, and the retrieval can be performed in reasonable times (it is just an *edit distance*).

Further work includes generating “cleaner” texture scores (for issues on the front-end, see [2]), and optimizing the computation of our *generalized edit distance*. The scheme still has to be tested on a large corpus of tunes and genres, but we believe that these results show the relevance of this alternative approach to Music Retrieval.

## 6. ACKNOWLEDGMENTS

Thanks to Maxime Crochemore from Institut Gaspard Monge, Marne-La-Vallee, France, and to Yoan J. Pinzon from King’s College London, UK, for discussions on string matching. This work has been partially supported by the British Council, UK

## 7. REFERENCES

- [1] Foote J., “ARTHUR: Retrieving Orchestral Music by Long-Term Structure”. In Proc. 1<sup>st</sup> ISMIR, October 2000.
- [2] Aucouturier, J-J, and Sandler, M., “Segmentation of Musical Signals Using Hidden Markov Models”. In Proc. AES 110<sup>th</sup> Convention, May 2001.
- [3] Rabiner, L.R. and Juang, B.H., “Fundamentals of speech recognition”. Prentice-Hall 1993
- [4] Rabiner, L., “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”. In Proc. IEEE, vol. 77, no. 2, 1989.
- [5] Crochemore, M., and Rytter, W., “Text Algorithms”. Oxford University Press, 1994.
- [6] Baker, B.S., “Parameterized Pattern Matching: Algorithms and Applications”, to appear in J. Comput. Syst. Sci