FINDING REPEATING PATTERNS IN ACOUSTIC MUSICAL SIGNALS : APPLICATIONS FOR AUDIO THUMBNAILING.

JEAN-JULIEN AUCOUTURIER[1], **MARK SANDLER**[2]

[1] *Sony Computer Science Laboratory, 6 rue Amyot, 75005 Paris, France*
jj@csl.sony.fr
[2] *Dep. of Electronic Engineering, Queen Mary University of London, Mile End Road, London E1 4NS, UK*
mark.sandler@elec.qmul.ac.uk

Finding structure and repetitions in a musical signal is crucial to enable interactive browsing into large databases of music files. Notably, it is useful to produce short summaries of musical pieces, or "audio thumbnails". In this paper, we propose an algorithm to find repeating patterns in an acoustic musical signal. We first segment the signal into a meaningful succession of timbres. This gives a reduced string representation of the music, the texture score, which doesn't encode any pitch information. We then look for patterns in this representation, using two techniques from image processing: Kernel Convolution and Hough Transform. The resulting patterns are relevant to musical structure, which shows that pitch is not the only useful representation for the structural analysis of polyphonic music.

INTRODUCTION

While listening to music, one often notices repetitions and recurrent structures. This is true for many different kinds of music: many 19th-century Europe compositions are built from one or several motives that are repeated and transformed (as studied e.g. by paradigmatic analysis [1]); most of modern popular music use the verse/chorus structure, and an instrument solo often answer to instrumental introduction and coda; "classic jazz", from "New Orleans" style to "Be-bop" is based on the repeated exposition of a theme, and improvisations around that theme; traditional folk music, e.g. the Celtic traditions from Ireland or Brittany, France, only uses a few different themes and phrases, with a great number of expressivity variations (timing, alterations, instrumentation).

The automatic discovery of patterns/motives/refrains in a piece of music has a number of applications in the context of large musical databases. Notably, in the framework of the European project Cuidado (*Content-based Unified Interfaces and Descriptors for Audio and Music Databases available Online*), we are focusing on :

- Retrieval and indexing : It is believed that content-based music retrieval should rely on patterns rather than on the whole score [2], and take account of musical structure [3]. For example, for some genres of music such as traditional folk, databases should be indexed by themes rather than by titles: most songs are built by concatenating old melodic themes taken from a common "repertoire".

- Browsing into a song : Pattern induction can be built into an enhanced music player, to allow intelligent fast-forward: the user can move automatically to the next chorus, to the guitar solo, or the next occurrence of what's currently being played.

- Audio-thumbnailing (or *abstracting*, or *summarizing*): The idea is to provide the user with the main characteristics of a title without playing it entirely. This would allow for instance a faster search among a set of titles, possibly the ordered result set of similarity search ([4]). One strategy to extract such a summary is to select the most reoccurring pattern in the song. Bartsch and Wakefield in [5] argue that, in the context of popular music, this amounts to picking up the chorus, which is likely to be recognized or remembered by the listener. Another possibility that we are currently investigating in the Cuidado framework, is to create a composite summary of a song by concatenating one occurrence of each small-scale pattern. The resulting "compressed" song contains only example of each of its most interesting phrases and structures, and is likely to convey the global "flavor" of the song.

There have been very many applications of pattern processing algorithms on musical strings. However, most of the proposed algorithms work from a transcription of music, or at least a MIDI-like sequence of notes, chords, etc They all assume (or rather capitalize on) a preliminary stage that could convert raw audio to such sequences of pitches. In this symbolic space, most of them rely on the computation of *edit distances* between patterns (complete overviews can be found in [6] and [7]). Many variants on edit-distances are proposed, such as using costs for edit operations that depend on musical rules (for instance, substituting a note by its octave should be cheaper than substituting it by a sixth minor) [8], or depend on

context [6]. To avoid the computation of $O(nm)$ distances between all possible sub-strings of the analyzed string of music, Hsu in [2] uses sub-trees, and Rolland [8] recursively computes the distances between large patterns from the already computed distances between smaller patterns.

To our knowledge, only two authors have previously addressed the problem of pattern induction in an *acoustic signal*, both in the context of music thumbnailing. Logan and Chu in [9] propose a method to find the "best" pattern in a song. First, each frame of the signal is labelled according to its spectral properties (or "timbre", modelled by Mel-Frequency Cepstrum Coefficients). The most reoccurring label within the song is then identified as the song's "global timbre". Finally, they select as a "key phrase" a section of the song which has the same global timbre as the whole song. Bartsch and Wakefield in [5] also offer to find only the "best pattern". Rather than reducing the signal to its timbre, and getting rid of most of the pitch and harmony information, they propose a reduced spectral representation of the signal that encodes its harmonic content (the *chromagram*). By autocorrelation, they identify the extract of the song whose "harmonic structure" re-occurs the most often.

In this paper, we describe a novel technique to extract meaningful patterns in an acoustic musical signal. Like in [9], it focuses on timbre rather than pitch or harmonic information : it is based on a segmentation of the signal into sections of constant timbre (section 1). But then, while Logan and Chu perform a *static* clustering of these timbres, we look at their *dynamics* over time: the segmentation gives a simple string representation of the music, the *texture score*, and we perform approximate pattern induction on this texture score (section 2). Just as [5] matches "successions of harmonic contents", here we match "successions of timbres". Section 3 will discuss the pros and cons of this approach.

1. THE TEXTURE SCORE 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 [10]. 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 Cepstrum Coefficients ([11]). A hidden Markov model (HMM, see [12]) 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.

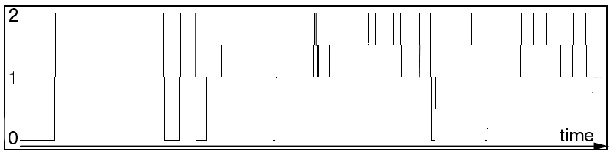

Figure 1: Texture Score of a 60's French song by Bourvil ([13]). State 0 is {silence}, state 1 is {voice + accordion + accompaniment} and state 2 is {accordion + accompaniment}

The "texture score" representation is just the succession over time of the textures learned by the model (Fig.1). It 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 $\ldots 11221333441 \ldots$ out of the alphabet {1,2,3,4}. The texture score shows a lot about the structure of the song: on Fig.1, the instrumental introduction appears very clearly, as well as the periodicities of the verse. In [14], the authors have used this representation in a Music Information Retrieval perspective to match different performances of the same song. In this paper, we offer to use the texture score to find repeating patterns in a song.

2. A GRAPHICAL FRAMEWORK FOR PATTERN INDUCTION

In order to discover patterns in the texture score string, we do not rely on dynamic programming, but rather have developed our own algorithm inspired by two image processing techniques: kernel convolution, and line detection with the Hough Transform. To our knowledge, it is the first time this point of view has been taken for a one-dimensional string matching. As shown in this section, this approach is less precise than edit distances, but still robust enough for most problems, very intuitive and fast enough for our needs. Although running time comparisons have not been done with the fastest implementations of dynamic programming ([8]), our graphical approach, notably the kernel algorithm, runs faster than most of our attempts at conventional pattern induction.

2.1. Definitions

Let x be a string, of length m, over an alphabet Σ. We note x_i^t the sub-string of x of length t, starting at index i: $x_i^t = x_i x_{i+1} \ldots x_{i+t}$

A length $n (n < m)$ string y is an *exact pattern* in x, if

there exist at least two indexes i,$j < m$, so that:

$$y = x_i^n \quad \text{and} \quad y = x_j^n \tag{1}$$

where i, and j are called the occurrences in x of the pattern y.
We define a string matching distance d on the set of all strings over Σ, such as an edit distance ([15]). A length $n (n < m)$ string y is then called an *approximate pattern* in x, if there exist at least two indexes i,$j < m$, so that:

$$d(y, x_i^n) \quad \text{and} \quad d(y, x_j^n) \quad \text{are not "too big"}, \tag{2}$$

where i, and j are called the occurrences in x of the pattern y.
Many criteria can apply for the "not to big" threshold, either absolute (as in δ-approximate matching as described by Crochemore [15]: no more than d errors), relative (no more than a certain percentage of the length of the pattern), or more complex (context-dependency as suggested by Cambouropoulos [6].

2.2. The exact matching problem

2.2.1. The correlation matrix

Our approach to pattern induction is based on a "correlation matrix". Let $x = x_1 x_2 \dots x_m$ be the string we want to analyze. The correlation matrix $\mathcal{M}_x$ is defined by:

$$\begin{aligned} \forall (i,j) \in [1,m]^2, \quad \mathcal{M}_x(i,j) = 1 \quad \text{if } x_i = x_j \\ \text{and} \quad \mathcal{M}_x(i,j) = 0 \quad \text{if } x_i \neq x_j \end{aligned} \tag{3}$$

Fig. 2 shows the correlation matrix $\mathcal{M}_x$ for a given string $x = 0001110001112211\dots$ The string x has been aligned on each axis for a more convenient reading. The matrix is obviously symmetric, thus only the upper half needs be considered.

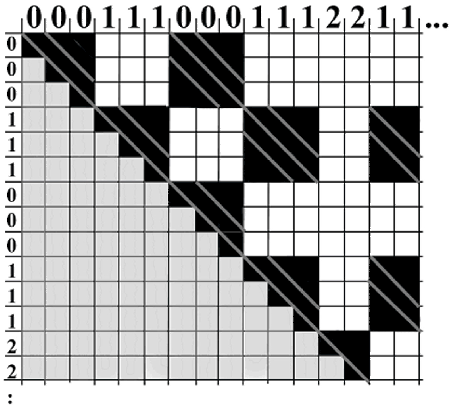

Figure 2: The correlation matrix for the string $0001110001112211\dots$. Cells equal to one appear in black, and the patterns appear as diagonals.

It appears clearly that diagonal segments in the matrix denote exact occurrences of a pattern in the string x. The longer the diagonal segment is, the longer the pattern. In Fig. 2, the longest diagonal segment corresponds to the alignment of $x_7^6 = 000111$ with $x_1^6 = 000111$. Therefore, finding exact patterns of any length in x just amounts to finding diagonal segments in $\mathcal{M}_x$.
The advantages of this representation will become only significant in the approximate matching case. However, it already provides a useful framework for "book-keeping": it is easy to discard trivial patterns, and to cluster all the segments into meaningful patterns, as we see in the next two sub-sections.

2.2.2. Discarding trivial patterns

There are a lot of patterns that one can find in a string, the majority of them being trivial given the knowledge of a minority of relevant ones. Let x be the analyzed string, y be a pattern, and x_i^t an occurrence of y in x.

- The *occurrence* x_i^t is called *trivial* if there exist a pattern z so that the occurrence x_i^t is logically deducible given the knowledge of an occurrence of z. For example, in Fig. 2, the occurrence x_8^2 of $y = 00$ is trivial, since there exist a pattern $z = 000111$ occurring in x_7^6. (We say that x_8^2 is trivial given x_7^6)

- A *pattern* is called *trivial*, if all of its occurrences are trivial. For example, if $y = 00$ only occurs as a substring of larger occurrences (such as occurrences of $z = 000111$), then the pattern is trivial.

Discarding trivial occurrences with the correlation matrix representation is easy. One just has to observe that for an occurrence x_i^t (i.e. a diagonal segment), all trivial occurrences x_j^s given x_i^t are found in a "square-neighborhood" of x_i^t (i.e. they are shorter and parallel diagonal segments that form a square around x_i^t). The black squares appearing in Fig. 2 demonstrate this phenomenon.

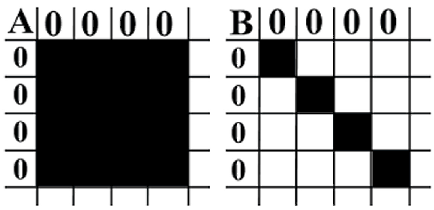

Figure 3: Illustration of "square reduction": all trivial diagonal segments in A are discarded in B

Some of these trivial occurrences are not stored in the first stage as we only store the biggest possible diagonals: a length-3 diagonal included in a length-5 diagonal is not

picked as a candidate for an occurrence. All the others can be discarded through the procedure of "square reduction" illustrated in Fig. 3
Therefore, to discard trivial *patterns* :

1. Discard all trivial *occurrences* of every pattern through "square-reduction" as demonstrated in Fig. 3.
2. If a pattern doesn't have any occurrence left, discard it as trivial.

2.2.3. Managing non-trivial patterns

Two other procedures can then be applied to cluster and organize the remaining patterns (i.e. patterns who have a least one non-trivial occurrence), so that the final list of patterns is compact and easy to read:

1. For each non-trivial pattern, recover all its trivial occurrences that have been discarded in the square reduction stage, or that are substring of another pattern's occurrence (see Fig. 4).

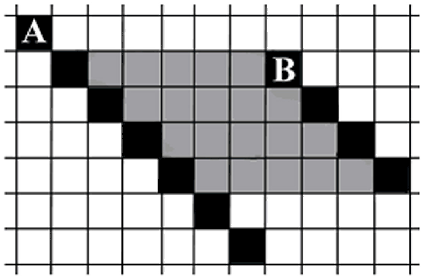

Figure 4: Recovering of a trivial occurrence included in A of a non-trivial pattern B

2. Coherently link all occurrences of the same pattern: if a diagonal is found that aligns x_i^t and x_j^t, and another diagonal aligns x_j^t and x_k^t, then x_i^t,x_j^t and x_k^t are occurrences of the same pattern (see Fig. 5).

2.3. The approximate matching problem

Most of the time, pattern induction must allow approximate matching. When using a score notation, patterns can have altered or transposed pitches, and show timing variations. Similarly, when using our texture score, the timing of the succession of timbres can vary from one occurrence to another, and the system must also account for noise in the segmentation.
In "classic" string matching, such approximate occurrences of a pattern can be modelled using 3 basic operations on the characters of the string: substitution, insertion and deletion ([15]). Each of these three operations has a graphical alter ego on the correlation matrix, which distorts

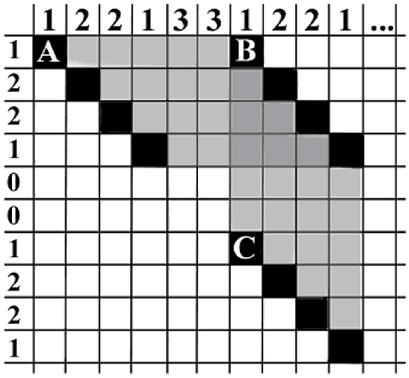

Figure 5: Linking occurrences A, B and C as occurrences of the same pattern

the diagonal segments that would occur in case of exact matching. Fig. 6 shows the three types of distortions that can occur in approximate occurrences.

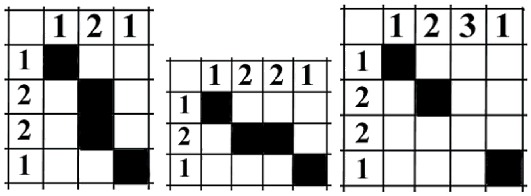

Figure 6: Diagonal distortion due to the three basic edit operations in approximate matching: (from left to right) deletion, insertion and substitution.

Hence, in our graphical framework, finding approximate occurrences amounts to finding "approximate diagonal segments", i.e. segments that are shifted downwards, leftwards, and/or interrupted. These distortions on quasi-diagonal segments can be viewed as noise -a very specific noise with known properties-, and we propose to use two image processing techniques to cope with this noise: kernel convolution, and line detection via the Hough Transform.

2.4. Kernel Convolution

2.4.1. Principle

The idea behind kernel convolution is to process the image so that we can find approximate patterns using the tools of exact matching. Thus, we want to "diagonalize" the approximate, distorted diagonal segments, by blurring together all the interrupted bits. Fig. 7 shows such an approximate diagonal, and what it would look like if blurred into a contiguous segment.
Figure 7 shows that such a "diagonalization" creates difficulties to localize the occurrence precisely: the blurred

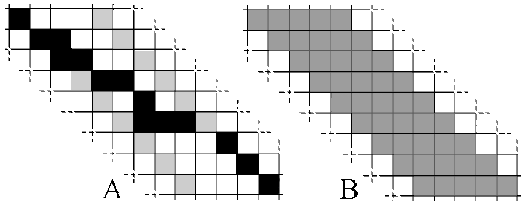

Figure 7: An approximate diagonal (A) with deletions, insertions and substitutions is blurred into a contiguous diagonal (B).

diagonal segment has a width. This phenomenon translates the fact that several alignments can be found between a pattern and its approximate match, as with an edit distance ([15]). In Fig. 7B, there are 5 possible alignments.

2.4.2. The Kernel

To achieve this blurring, we smooth the matrix with a specially shaped convolution kernel. The kernel consists mainly of a Gaussian distribution extruded over the diagonal, with a reinforced diagonal. To minimize side effect, this "Gaussian line" is smoothed using windows that taper towards zero at each end of the diagonal. The resulting shape is shown in Figure 8.

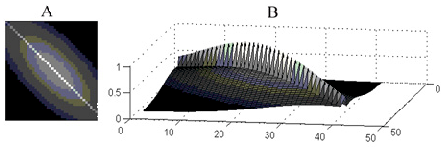

Figure 8: Top view (A) and 3D view (B) of the diagonal kernel used in approximate matching.

Convolving the matrix $\mathcal{M}_x$ with this kernel amounts to slide along each diagonal of $\mathcal{M}_x$ and integrate the neighboring pixels so that any "quasi-diagonal" is reinforced into an exact, contiguous segment. The diagonal dimension of the kernel compensates for substitutions (gaps in a diagonal segment), and its anti-diagonal dimension compensates for insertions and deletions (horizontal or vertical shifts in a diagonal segment). This convolution turns the black and white image of $\mathcal{M}_x$ into a gray-level picture, where the highest values denote the least noisy diagonal segments. A threshold is then applied to keep only the best occurrences, which are processed just as before, with square reduction, sharing and linking.

Figure 9 shows an approximate occurrence before and after kernel convolution. It appears that the "best diagonal" is easily extracted from the gray-level representation.

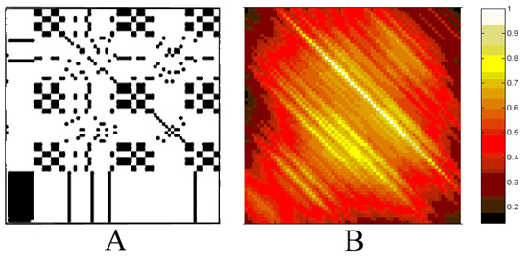

Figure 9: Detail of a correlation matrix with an approximate occurrence before (A) and after convolution with the diagonal kernel (B)

2.4.3. About the size of the kernel.

The choice of the best size for the diagonal kernel mainly depends on the level of approximation that we want the system to account for: a larger kernel will compensate for more distortion than a small one. The size of the kernel is roughly equal to the maximum number of edit operations it can compensate in a pattern.
However, it is not easy to set a relative threshold with the kernel method, as the size of the kernel doesn't depend on the size of the patterns occurring in the matrix.
Another parameter that can be tuned is the width of the Gaussian shape, i.e. its standard deviation. For our experiments, it's been fixed to a quarter of the kernel's side.

2.5. The Hough Transform

2.5.1. Principle

In this approach, we stop looking for diagonal segments, but rather for segments of any slope. We use a line detection tool, the Hough Transform, to find the best straight lines that go through the black pixels of the correlation matrix, and then find the occurrences along these lines. The approach is illustrated in Figure 10, with the same example as in Figure 7.

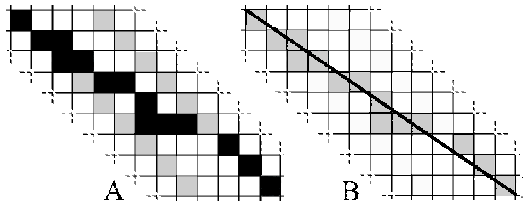

Figure 10: An approximate diagonal (A) with deletions, insertions and substitutions processed by the Hough Transform (B).

2.5.2. The Transform

The Hough Transform is a widely established technique for detecting complex patterns of points in binary images,

notably straight lines. For a complete survey of its use in Computer Vision, see [16]. Our work is not the only one to use the Hough Transform for audio processing: notably, Townsend in [17] uses it to track formants in percussive sounds.

Let (x, y) be the coordinates of a pixel in an image. All lines that pass through this pixel obey to an equation of the form:

$$y = mx + c \tag{4}$$

for all values of $(m, c) \in \Re^2$. A given line going through (x, y) is thus entirely given by its coordinate (m_0, c_0) , and in the (m, c) space (also called "parameter space"), the set of all lines going through this given pixel (x, y) corresponds to all the points (m, c) so that:

$$c = -xm + y \tag{5}$$

This is a straight line on a graph of c against m. This mapping from a pixel (x, y) in the image space to an infinite line in the parameter space can be repeated for every pixel of the image, producing a parameter space for the whole image (Figure 11).

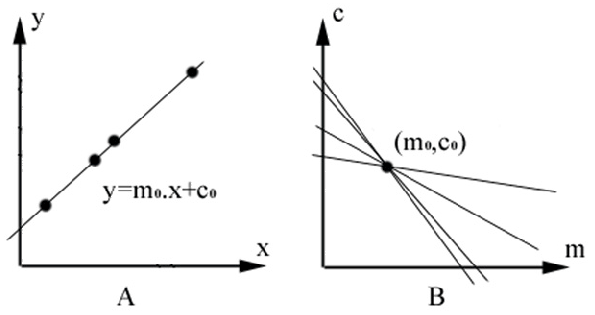

Figure 11: An image (A) and its corresponding parameter space (B): the point of intersection in the parameter space gives the common line that goes through the four points in picture A.

In figure 11, the point of intersection (m_0, c_0) in the parameter space (B) gives the parameters of a line that goes through every point of the image A : it is the common line we are looking for.

In practice, not all pixels in an image lie on the same line, and the parameter space thus looks more complicated than in Figure 11. The determination of the intersection(s) in the parameter space becomes a local maximum detection problem. Each local maximum (m_0, c_0) correspond to a line in the original image. Figure 12 shows the result of the Hough Transform on a "real-world" correlation matrix. The two arrows in Figure 12B pinpoint two relevant local maxima on the parameter space. The local maxima are looked for in an area corresponding to slopes between 40 and 50 degrees, as we only want "quasi-diagonals", and not horizontal lines, say. The two gray lines in Figure 12A show the corresponding lines on the correlation matrix. Contrary to the kernel method, the slopes are not necessarily 45 degrees.

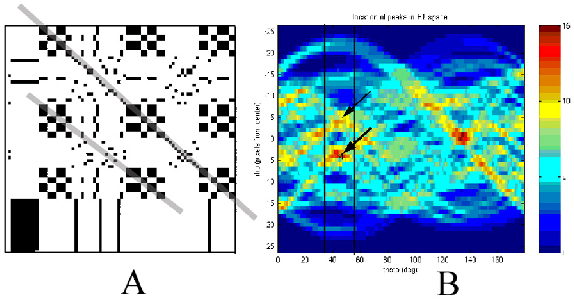

Figure 12: A detail of a correlation matrix (A) and the corresponding parameter space (B): The two arrows in (B) pinpoint relevant local maximums and gray lines in (A) show the corresponding occurrences.

2.5.3. Advantages of The Hough Transform:

For our problem of pattern induction, the Hough Transform has three interesting properties:

1. It is very good at handling bad or missing data points. Spurious data doesn't contribute much to the parameter space, and thanks to the parameterization, the pixels which lie on the line need not all be contiguous. Thus, it is very effective to detect our distorted and interrupted quasi-diagonal segments.

2. Better localization of the patterns: As seen previously, Kernel Convolution blurs several bits of diagonals into a bigger and thicker one, which creates several equivalent alignments, and thus an ambiguous localization of the patterns. On the contrary, the Hough Transform find the one best alignment, without trying to fit it to a 45 degree slope. Consequently, less redundant patterns are found, and the results of the algorithm are clearer and more compact to read.

3. Easier to set the maximum rate of approximation allowed: Contrary to Kernel Convolution, the Hough Transform allows the user to specify a relative threshold (i.e. a max number of errors which is a fixed percentage of the patterns' length), which is much more realistic and efficient in the context of pattern induction. There is a direct relation between the slope of the detected lines and the number of errors in the pattern: the more errors, the more different the slopes are from a diagonal's 45 degrees. Let N be the length of a pattern, X be the number

of errors in its occurrence, and q the angle deviation from a 45-degree diagonal. In the worst case (all errors are deletions, or all are insertions), the length-M diagonal segment is shifted by X steps.

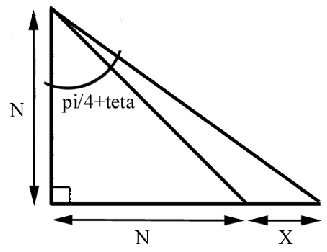

Figure 13: Deviation from the diagonal when a N-length pattern occurs with X errors

From Figure 13, we can write:

$$\tan\frac{\pi}{4}+\theta=\frac{N+X}{N} \tag{6}$$

and therefore:

$$\theta=\arctan\frac{X}{2N+X} \tag{7}$$

We define the error rate as:

$$\rho=\frac{X}{N} \tag{8}$$

Equation 7 then becomes:

$$\theta=\arctan\frac{\rho}{2+\rho} \tag{9}$$

Equation 9 thus gives a direct relation between the error rate and the maximum deviation from 45 degrees that the segments can have. It is therefore sufficient to look for local maximum in the accumulator for values of the slope m between $\frac{\pi}{4}-\theta$ and $\frac{\pi}{4}+\theta$. Such boundaries have been used in Figure 12.

2.5.4. Disadvantages of The Hough Transform:

Although the Hough Transform has a number of advantages over the kernel method, it also has two practical disadvantages:

1. The transform gives an infinite line as expressed by the pair of m and c values, rather than a finite line segment with two well defined endpoints. In fact, it does not even guarantee that there exists any finite length line in the image (i.e. a sufficient number of contiguous pixels). The existence of a finite line segment (i.e. a valid occurrence) must therefore be verified. The verification is performed by scanning the pixels along the line and checking whether they meet certain criteria, like maximum gap between pixels.
2. Computationally, the Hough Transform is more demanding than the simple convolution used in the kernel method. Notably, it involves searching for a local maximum. A way to reduce the running time is to compute the transform by windows, but further routines are necessary to manage overlapping segments.

In a nutshell, the Hough Transform is a powerful tool for pattern discovery: it is precise, well adapted to time-warping (i.e. deviation from 45 degree), and easy to parameterize (thanks to the relation between the maximum slope and the error rate). However, its practical use is less straightforward than Kernel Convolution, which uses the same routines as in the exact matching case. The Kernel method is less precise and less convenient to parameterize (e.g. choice of the kernel's size), but still yields exploitable results, and is considerably faster.

3. RESULTS

We have applied our graphical algorithm to discover patterns on several "texture scores" obtained with our segmentation algorithm reviewed in section 1.

3.1. Pattern induction on Bourvil's "C'était bien"

A first attempt using just the *exact* matching algorithms yields no useful results, the patterns being either too short (a few 30 ms frames) or not meaningful (two seconds of a quasi-constant texture occurring during an instrumental solo). This confirms the importance of approximate pattern induction.

The *approximate* pattern analysis on the texture score of Bourvil's song "C'était bien" [13] reveals a lot of the long-term structure of the tune. Notably, our algorithm discovers a very big approximate pattern that correspond to the alternation of verse, chorus and solo instrument. This unit in itself, which length is about the third of the whole song's, would provide a good summary of the tune. However, the results are even more interesting when we look at shorter patterns, which correspond to phrases within a verse or a chorus. On the next pages, we present a very convincing example of such a pattern. Its length is relatively small, about 3 seconds. It occurs 15 times during the song, 5 times in each occurrence of the verse/chorus unit. Figure 14 presents five of its occurrences (the first five in the first chorus), and Figures 15 to 19 show a transcription by the author of the corresponding music.

The state sequences shown in Fig. 14 have the same labelling than in section 1: state 1 is silence, state 2 is {voice+accompaniment} and state 3 is {accompaniment}.

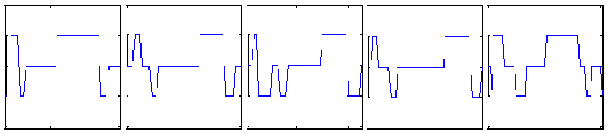

Figure 14: Five occurrences of a pattern in Bourvil's texture score

In the transcriptions shown in Figures 15 to 19, the upper staff corresponds to the vocal score, and the two bottom staffs correspond to the accompaniment: accordion, and bass. The drum track has not been transcribed, as it doesn't influence the segmentation very much.

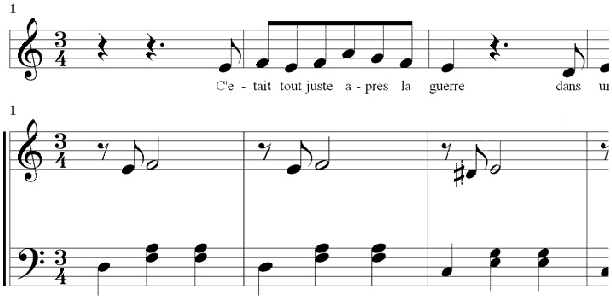

Figure 15: transcription of the first occurrence of the pattern

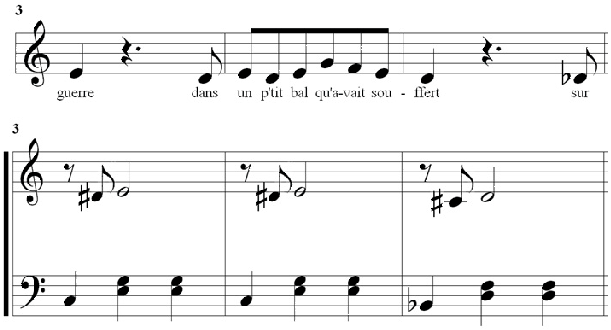

Figure 16: transcription of the second occurrence of the pattern

We can see from the transcriptions in Figure 15–19 that these 5 occurrences correspond to the same sequence of scale degrees $(2-3-2-3-5-4-3-2)$, but diatonically transposed to 5 levels, harmonized in Dm, C, Bb, F, Eb. Classic pattern induction algorithms would deal with such a pitch similarity by using musical rules to account for transposition, or by just looking at musical contour. In our case, this similarity of the pitches can't be assessed from the texture score, since it hides all pitch information within the textures. The algorithm thus has discovered some similarity based something else: structure. These

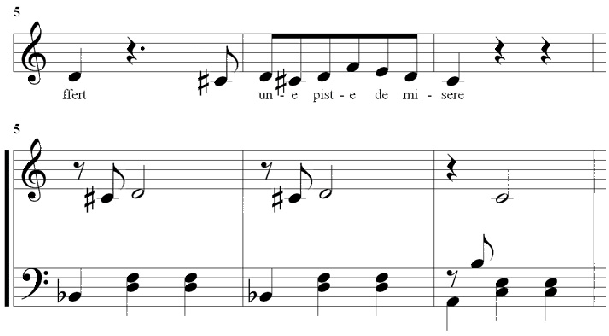

Figure 17: transcription of the third occurrence of the pattern

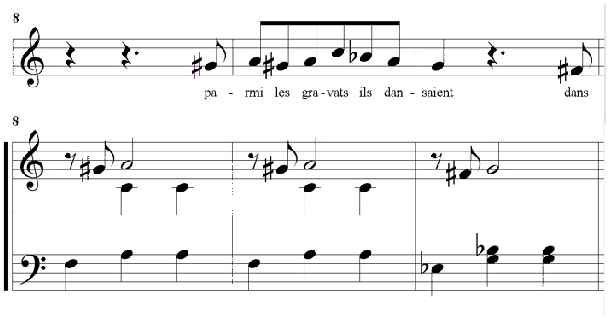

Figure 18: transcription of the fourth occurrence of the pattern

occurrences have the same succession of textures. We may even say that this pitch similarity via transposition has been discovered because we don't account for pitches: we've discovered a similarity by looking at what was the same (texture timing) and not looking at what was different (pitches).

Note that the variations between the occurrences, such as the duration of the textures, correspond to variations of timing and expressivity on the same phrase. This is especially clear about the frames of silence (texture 1) which reveal short pauses between sung words or in the accompaniment.

It is remarkable that melodic phrases and texture timing be so closely correlated, and this suggests that a pitch transcription may not be the only useful notation to understand music. In the context of music processing, this opens the way for alternative, more abstract representations of polyphony, which are easier to generate from raw data, without having to separate sources. The texture score appear to be a good example of such a representation.

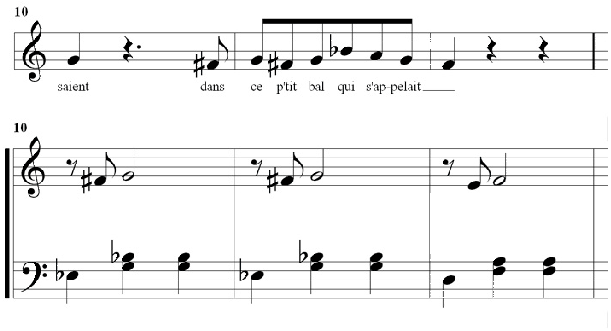

Figure 19: transcription of the fifth occurrence of the pattern

3.2. Discussion

3.2.1. Advantages of the approach

- String-matching vs clustering : Our approach is based on timbre properties of the signal, like Logan's ([9]). However, instead of clustering sections of quasi-constant timbre, we match quasi-constant *successions* of timbres. This gives much more insight into the piece's musical phrases. We are notably able to differentiate patterns which yet have the same "global timbre". One important conclusion that can be drawn from our results is that "timbre" is not just useful to give a global and somewhat blurry snapshot of a song, but allows a rather precise analysis of musical structure.

- String-matching vs autocorrelation : In our approach, we use string-matching techniques on a symbolic representation learned from the data : the texture score. This is likely to be more robust for approximate matching than simple signal correlation used by Bartsch and Wakefield ([5]), which "fails when the chorus is repeated with some change in the musical structure of the repeat". Moreover, the tools of approximate string matching (either our homemade graphical algorithm or classical edit-distance based algorithms) are highly customable, and may even include musical rules ([6]), which allows more control on the patterns we want to select. Such symbolic algorithms also facilitates the book-keeping of a hierarchy of patterns, which is necessary if we want to build a composite summary of the piece by concatenating different patterns.

3.2.2. Disadvantages and Future Work

- Pitch patterns : One originality of our approach is that it doesn't rely on pitch or harmonic content, but rather on timbres. However, some patterns can't be found by just looking at timbres (e.g. when there is only one texture, like in a solo piano piece). As Bartsch and Wakefield's chromogram seems to be a promising representation of the harmonic content in a song, we plan to adapt our algorithm to their encoding: First a model-based segmentation in the chromogram space, and then string-matching to find patterns. It is likely that both approaches (timbre and harmonic content) will be complementary.

- Objective evaluation : It is hard to evaluate objectively the relevance of our approach, although the numerous experiments that we made show a high correlation between melodic patterns and texture timing. Using the same algorithm on the chromogram would provide a framework to evaluate the relevance of "timbre patterns", possibly by measuring the overlap between both sets of patterns. We also plan to use the patterns found by our algorithm to create music thumbnails, and conduct a statistical subjective evaluation on large databases of songs in the framework of the Cuidado project. Both [5] and [9] suggest some evaluation methods, and a music-cognition point of view can be found in [18].

- Changes in instrumentation : There are patterns in music that can't be found from the texture score. For example, in Bourvil's song, the vocal part in the chorus answers a phrase that is exposed in the introduction by the accordion. The notes are very similar, and this pattern would probably be picked by an algorithm that works from a transcription. However, the textures are completely different, and thus can't be matched. This problem is difficult to solve from a signal approach, and is also a caveat in Bartsch's system ([5]). With our texture score representation, one could imagine matching patterns with a random permutation of the textures (e.g. 00111001122 is matched with 11222112200). Such an algorithm has been used by the authors in [14] to match different versions of the same song. However, this would dramatically increase the number of matches, and it is unclear whether they would all be relevant. Moreover, the complexity of the matching process would be much higher.

4. CONCLUSION

Patterns in music are very important both to musicologists and to computer scientists. In this paper, we have presented a graphical framework for quick pattern induction in strings. We have investigated two techniques from image processing: Kernel Convolution and Hough Transform. Both have pros and cons, which we have discussed. We have applied these algorithms to find patterns on a

timbre representation of music, the texture score. We have shown that the resulting patterns are relevant to musical structure, and that there often is a correlation between texture timing and melodic similarity. This highlights that a pitch transcription may not be the only useful representation for the structural analysis of polyphonic music.

REFERENCES

[1] J.-J. Nattiez. *Fondements d'une sémiologie de la musique*. Union Générale d'Éditions, Paris, 1975.

[2] J.-L. Hsu, C.-C. Liu, and A. Chen. Efficient repeating pattern finding in music databases. In *Proceedings of the Conference in Information and Knowledge Management*, 98.

[3] M. Melucci and N. Orio. The use of melodic segmentation for content-based retrieval of musical data. In *Proceedings of the International Computer Music Conference, Beijng, China*, 1999.

[4] G. Tzanetakis and P. Cook. Audio information retrieval (air) tools. In *Proc. International Symposium on Music Information Retrieval*, 2000.

[5] M. Bartsch and G. Wakefield. To catch a chorus: Using chroma-based representations for audio thumbnailing. In *Proc. 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics*, 2001.

[6] E. Cambouropoulos, T. Crawford, and C. Iliopoulos. Pattern processing in melodic sequences: Challenges, caveats and prospects. *Computers and the Humanities*, 34(4), 1999.

[7] T. Crawford, C.S. Iliopoulos, , and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11:71–100, 1998.

[8] P.-Y. Rolland. Flexpat: A novel algorithm for musical pattern discovery. In *Proc. of the XII Colloquium in Musical Informatics, Gorizia, Italy*, 1998.

[9] B. Logan and S. Chu. Music summarization using key phrases. In *Proc. International Conference on Acoustics, Speech and Signal Processing*, 2000.

[10] J.-J. Aucouturier and M. Sandler. Segmentation of musical signals using hidden markov models. In *Proc. 110th Convention of the Audio Engineering Society*, 2001.

[11] L.R. Rabiner and B.H. Juang. *Fundamentals of speech recognition*. Prentice-Hall, 1993.

[12] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, vol. 77(2), 1989.

[13] Bourvil. C'était bien (le petit bal perdu), 1961. Lyrics: R. Nyel, Music: G. Verlor, Editions Bagatelle.

[14] J.-J. Aucouturier and M. Sandler. Using long-term structure to retrieve music : Representation and matching. In *Proceedings of the 2nd International Symposium on Music Information Retrieval ISMIR*, 2001.

[15] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12:244–250, 1981.

[16] V.F. Leavers. *Shape detection in Computer Vision Using the Hough Transform*. Springer-Verlag, 1992.

[17] M. Townsend. *Analysis of Percussive Sounds Using Linear Predictive Coding*. Ph.d. thesis, King's College, University of London, London, U.K., 1994.

[18] D. Huron. Perceptual and cognitive applications in music information retrieval. In *Proceedings of International Symposium of Music Information Retrieval*, 2000.