Pattern Discovery Techniques for Music Audio

Roger B. Dannenberg and Ning Hu
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
1-412-268-3827

{rbd,ninghu}@cs.cmu.edu

ABSTRACT

Human listeners are able to recognize structure in music through the perception of repetition and other relationships within a piece of music. This work aims to automate the task of music analysis. Music is "explained" in terms of embedded relationships, especially repetition of segments or phrases. The steps in this process are the transcription of audio into a representation with a similarity or distance metric, the search for similar segments, forming clusters of similar segments, and explaining music in terms of these clusters. Several transcription methods are considered: monophonic pitch estimation, chroma (spectral) representation, and polyphonic transcription followed by harmonic analysis. Also, several algorithms that search for similar segments are described. These techniques can be used to perform an analysis of musical structure, as illustrated by examples.

1. INTRODUCTION

Digital sound recordings of music can be considered the lowest level of music representation. These audio representations offer nothing in the way of musical or sonic structure, which is problematic for many tasks such as music analysis, music search, and music classification. Given the current state of the art, virtually any technique that reveals structure in an audio recording is interesting. Techniques such as beat detection, key detection, chord identification, monophonic and polyphonic transcription, melody and bass line detection, source separation, speech recognition, and instrument identification all derive some higher-level information from music audio. There is some hope that by continuing to develop these techniques and combine them, we will be better able to reason about, search, and classify music, starting from an audio representation.

In this work, we examine ways to discover patterns in music audio and to translate this into a structural analysis. The main idea is quite simple: musical structure is signaled by repetition. Of course, "repetition" means similarity at some level of abstraction above that of audio samples. We must process sound to obtain a higher-level representation before comparisons are made, and must allow approximate matching to allow for variations in performance, orchestration, lyrics, etc. In a number of cases, our techniques have been able to describe the main structure of music compositions.

We have explored several representations for comparing music. Monophonic transcription can be used for music where a single voice predominates (even in a polyphonic recording). Spectral frames can be used for more polyphonic material. We have also experimented with a polyphonic transcription system.

For each of these representations, we have developed heuristic algorithms to search for similar segments of music. We identify pairs of similar segments. Then, we attempt to simplify the potentially large set of pairs to a smaller set of clusters. These clusters identify "components" of the music. We can then construct an explanation or analysis of the music in terms of these components. The goal is to derive structural descriptions such as "AABA."

We believe that the recognition of repetition is a fundamental activity of music listening. In this view, the structure created by repetition and transformation is as essential to music as the patterns themselves. In other words the structure AABA is important regardless of what A and B represent. At the risk of oversimplification, the first two A's establish a pattern, the B generates tension and expectation, and the final A confirms the expectation and brings resolution. Structure is clearly important to music listening. Structure can also contribute expectations or prior probabilities for other analysis techniques, such as transcription and beat detection, where knowledge of pattern and form might help to improve accuracy. It follows that the analysis of structure is relevant to music classification, music retrieval, and other automated processing tasks.

2. RELATED WORK

It is well known that music commonly contains patterns and repetition. Any music theory book will discuss musical form and introduce notation, such as "AABA," for describing musical structures. Many researchers in computer music have investigated techniques for pattern discovery and pattern search. Cope [4] searches for "signatures" that are characteristic of composers, and Rolland and Ganascia describe search techniques [21]. Interactive systems have been constructed to identify and look for patterns [24], and much of the work on melodic similarity [9] is relevant to the analysis of music structure.

Simon and Sumner wrote an early paper on music listening and its relationship to pattern formation and memory [23], proposing that we encode melody by referencing patterns and transformations. This has some close relationships to data compression, which has also inspired work in music analysis and generation. [11] Narmour describes a variety of transformative processes that operate in music to create structures that listeners perceive. [18]

A fundamental idea in this work is to compare every point of a music recording with every other point. This naturally leads to a matrix representation in which row *i*, column *j* corresponds to the similarity of time points *i* and *j*. A two-dimensional grid to compute and display self-similarity has been used by Wakefield and Bartsch [1] and by Foote and Cooper [8].

Mont-Reynaud and Goldstein proposed rhythmic pattern discovery as a way to improve music transcription. [16] Conklin and Anagnostopoulou examine a technique for finding significant exactly identical patterns in a body of music. [3] A different approach is taken by Meek and Birmingham to search for commonly occurring melodies or other sequences. [14]

3. PATTERN SEARCH

In this section, we describe the general problem of searching for similar sections of music. We assume that music is represented as a sequence s_i, $i = 0 \ldots n-1$. A *segment* of music is denoted by a starting and ending point: (i, k), $0 \le i \le k < n$. Similar sections consists of two segments: $((i, k), (j, l))$, $0 \le i \le k < j \le l < n$. For convenience, we do not allow overlapped segments[1], hence $k < j$.

There are $O(n^4)$ possible pairs of segments. To compute a similarity function of two segments, one would probably use a dynamic programming algorithm with a cost proportional to the lengths of the two segments. This increases the cost to $O(n^6)$ if each pair of segments is evaluated independently. However, given a pair of starting points, i,j, the dynamic programming alignment step can be used to evaluate all possible pairs of segment endpoints. There are $O(n^2)$ starting points and the average cost of the full alignment computation is also $O(n^2)$, so the total cost is then $O(n^4)$. Using frame sizes from 0.1 to 0.25 seconds and music durations of several minutes, we can expect n to be in the range of 200 to 2000. This implies that a brute-force search of the entire segment pair space is will take hours or even days. This has led us to pursue heuristic algorithms.

In our work, we assume a distance function between elements of the sequence s_i. To compute the distance between two segments, we use an algorithm for sequence alignment based on dynamic programming. A by-product of the alignment is a sum of distances between corresponding sequence elements. This measure has the property that it generally increases with length, whereas longer patterns are generally desirable. Therefore, we divide distance by length to get an overall distance rating.

Typically there are many overlapping candidates for similar segments. Extending or shifting a matching segment by a frame or two will still result in a good rating. Therefore, the problem is not so much to find all pairs of similar segments but the locally "best" matches. In practice, all of our algorithms work by extending promising matches incrementally to find the "best" match. This approach reduces the computation time considerably, but introduces heuristics that make formal descriptions difficult. Nevertheless, we hope this introduction will help to explain the following solutions.

4. MONOPHONIC ANALYSIS

Our first approach is based on monophonic pitch estimation, which is used to transcribe music into a note-based representation. Notes are represented as a pitch (represented on a continuous rather than quantized scale), starting time, and duration (in seconds). The pitch estimation is performed using autocorrelation [20] and some heuristics for rejecting false peaks and outliers, as described in an earlier paper. [5]

We worked with a saxophone solo, "Naima," written and performed by John Coltrane [2] with a jazz quartet (sax, piano, bass, and drums). To find matching segments in the transcription, we construct a matrix M where $M_{i,j}$ is the length of a segment[2] starting at note i and matching a segment at note j.

4.1 Algorithm 1

The search algorithm in this case is a straightforward search of every combination of i, j such that $i < j$. For n notes, there are $n(n-1)/2$ pairs. The search proceeds only if there is a close match between pitch i and pitch j. Although we could use dynamic programming for note alignment [9, 22], we elected to try a simple iterative algorithm that attempts to extend the current pair of similar segments with additional matching notes to find the longest similar segments starting at i and j. The rules for extending segments consider note pitch and duration, and allow for (short) note deletions and consolidation [15].

If segment (i, k) matches (j, l), then in many cases, $(i + 1, k)$ will match $(j + 1, l)$ and so on. To eliminate the redundant pairs, we make a pass through the elements of M, clearing cells contained by longer similar segments. For example if (i, k) matches (j, l), we clear all elements of the rectangular submatrix $M_{i..k,j..l}$ except for $M_{i,j}$.

Finally, we can read off pairs of similar segments and their durations by making another pass over the matrix M. Although this approach works well if there is a good transcription, it is not generally possible to obtain a useful melodic transcription from polyphonic audio. In the next section, we consider an alternative representation.

5. SPECTRUM-BASED ANALYSIS

When transcription is not possible, a lower-level abstraction based on the spectrum can be used. We chose to use Wakefield's *chroma* because it seemed to do a good job of identifying similar segments in an earlier study where the goal was to find the chorus of a pop song. [1]

The chroma is a 12-element vector where each element represents the energy associated with one of the 12 pitch classes. Essentially, the spectrum "wraps around" at each octave and bins are combined to form the chroma vector. Distance is then defined as Euclidean distance between vectors normalized to have a mean of zero and a standard deviation of one.

The most important feature of a chroma representation is that the music is divided into equal-duration frames rather than notes. Typically, there will be hundreds or thousands of frames as opposed to tens or hundreds of notes. Matching will tend to be more ambiguous because the data is not segmented into discrete notes. Therefore, we need to use more robust (and expensive) sequence alignment techniques and therefore more clever algorithms.

5.1 Brute-Force Approach

At first thought, observing that dynamic programming computes a global solution from incremental and local properties, one might try to reuse local computations to form solutions to our similar segments problem. A typical dynamic programming step computes the distance at cell i,j in terms of cells to the left $(j-1)$, above $(i-1)$, and diagonal $(i-1, j-1)$. The value at i,j is:

$$\boldsymbol{M}_{i,j} = d_{i,j} + \min(\boldsymbol{M}_{i,j-1}, \boldsymbol{M}_{i-1,j}, \boldsymbol{M}_{i-1,j-1})$$

In terms of edit distances, we use $d_{i,j}$, the distance from frame i to frame j as either a replacement cost, insertion cost, or deletion cost, although many alternative cost/distance functions are possible within the dynamic programming framework. [10] Unfortunately, even if we precompute the full matrix, it does not help us in computing the distance between two segments because of initial boundary conditions, which change for every combination of i and j. As mentioned in the introduction, the best we can do is to compute a submatrix starting at i,j for every $0 \le i < j < n$. This leaves us with an $O(n^4)$ algorithm to compute the distance for every pair $((i, k), (j, l))$. To avoid very long computation times, we developed a faster, heuristic search.

5.2 Heuristic Search

We compute the distance between two segments by finding a path from i,j to k,l that minimizes the distance function. Each step of the path takes one step to the right, downward, or diagonally. In practice, similar segments are characterized by paths that consist mainly of diagonal segments because tempo variation is typically small. Thus we do not need to compute a

full rectangular array to find good alignments. Alternatively, we can compute several or even all paths with a single pass through the active matrix. This method is described here.

5.3 Algorithm 2

The main idea of this algorithm is to identify path beginnings and to follow paths diagonally across a matrix until the path rating falls below some threshold. The algorithm uses three matrices we will call distance (***D***), path (***P***), and length (***L***). ***D*** and ***L*** hold real (floating point) values, and ***P*** holds integers. ***P*** is initialized to zero so that we can determine which cells have been computed. If $\boldsymbol{P}_{i,j} = 0$, we say cell i,j is uninitialized. The algorithm scans the matrix along diagonals of constant $i+j$ as shown in Figure 1, filling in corresponding cells of ***D***, ***P***, and ***L***. A cell is computed in terms of the cells to the left, above, and diagonal. First, compute distances and lengths as follows:

$$d_h = \text{if } \boldsymbol{P}_{i,j-1} \neq 0 \text{ then } \boldsymbol{D}_{i,j-1} + d_{i,j}, \text{ else } \infty;\ l_h = \boldsymbol{L}_{i,j-1} + \sqrt{2}/2$$

$$d_v = \text{if } \boldsymbol{P}_{i-1,j} \neq 0 \text{ then } \boldsymbol{D}_{i-1,j} + d_{i,j}, \text{ else } \infty;\ l_v = \boldsymbol{L}_{i-1,j} + \sqrt{2}/2$$

$$d_d = \text{if } \boldsymbol{P}_{i-1,j-1} \neq 0 \text{ then } \boldsymbol{D}_{i-1,j-1} + d_{i,j}, \text{ else } \infty;\ l_d = \boldsymbol{L}_{i-1,j-1} + 1$$

The purpose of the infinity (∞) values is to disregard distances computed from uninitialized cells as indicated by ***P***. Now, let $c = \min(d_h/l_h, d_v/l_v, d_d/l_d)$. If c is greater than a threshold, the cell at i,j is left uninitialized. Otherwise, we define $\boldsymbol{D}_{i,j} = c$, $\boldsymbol{L}_{i,j} = l_m$, and $\boldsymbol{P}_{i,j} = \boldsymbol{P}_m$, where the subscript m represents the cell that produced the minimum value for c, either $(i,j-1)$, $(i-1,j)$, or $(i-1,j-1)$.

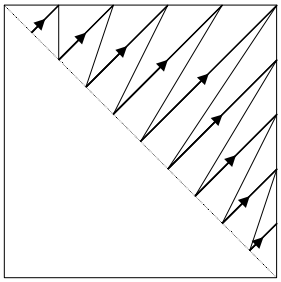

Figure 1. In Algorithm 1, the similarity matrix is computed along diagonals as shown.

As described so far, this computation will propagate paths once they are started, but how is a path started? When $\boldsymbol{P}_{i,j}$ is left uninitialized by the computation described in the previous paragraph and $d_{i,j}$ is below a threshold (the same one used to cut off paths), $\boldsymbol{P}_{i,j}$ is set to a new integer value to denote the beginning of a new path. We also define $\boldsymbol{D}_{i,j} = d_{i,j}$ and $\boldsymbol{L}_{i,j} = 1$ at the beginning of the path.

After this computation, regions of ***P*** are partitioned according to path names. Every point with the same name is a candidate endpoint for the same starting point. We still need to decide where paths end. We can compute endpoints by reversing the sequence of chroma frames, so that endpoints become starting points. Recall that starting points are uninitialized cells where $d_{i,j}$ is below a threshold. To locate endpoints, scan the matrix in reverse from the original order (Figure 1). Whenever a new path name is encountered, and the distance $d_{i,j}$ is below threshold, find the starting point and output the path. An array can keep track of which path names have been output and where paths begin.

6. POLYPHONIC TRANSCRIPTION

Polyphonic transcription offers another approach to similarity. Although automatic polyphonic transcription has rather high error rates, it is still possible to recover a significant amount of musical information. We use Marolt's SONIC transcription program [12], which transcribes audio files to MIDI files. SONIC does not attempt to perform source separation, so the resulting MIDI data combines all notes into a single track. Although SONIC was intended for piano transcription, we get surprisingly good results with arbitrary music sources. Transcriptions inevitably have spurious notes, so we reduce the transcriptions to a chord progression using the Harman program by Pardo [19]. Harman is able to ignore passing tones and other non-chord tones, so in principle, Harman can help to reduce the "noise" introduced by transcription errors.

After computing chords with Harman, we generate a sequence of frames s_i, $0 < i < n$, where each frame represents an equal interval of time and s_i is a set of pitch classes corresponding to the chord label assigned by Harman.

In our experiments with polyphonic transcription, we developed yet another algorithm for searching for similar segments. This algorithm is based on an adaptation of dynamic programming for computer accompaniment [6]. In this accompaniment algorithm, a score is matched to a performance not by computing a full $n \times m$ matrix but by computing only a diagonal band swept out by a moving window, which is adaptively centered on the "best" current score position.

6.1 Algorithm 3

To find similar segments, we will sweep a window diagonally from upper left to lower right as shown in Figure 2. When a match is found, indicated by good match scores, the window is moved to follow the best path. We need to decide where paths begin and end. For this purpose, we compute similarity (rather than distance) such that similarity scores increase where segments match, and decrease where segments do not match.

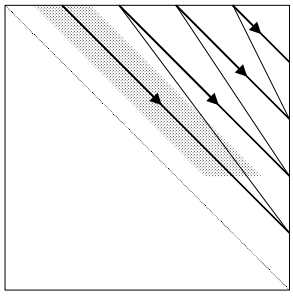

Figure 2. In Algorithm 2, the similarity matrix is computed in diagonal bands swept out along the path shown. The shaded area shows a partially completed computation.

An example function for similarity of chords is to count the number of notes in common minus the number of notes not in common. For chords A and B (sets of pitch classes), the similarity is:

$$\sigma(A, B) = |A \cap B| - |A \cup B - A \cap B|,$$

where $|X|$ is the number of elements in (cardinality of) set X. We will write $\sigma_{i,j}$ to denote $\sigma(s_i, s_j)$, the similarity between chords at frames i and j.

When we compute the matrix, we initialize cells to zero and store only positive values. A path begins when a window element becomes positive and ends when the window becomes zero again. The computation for a matrix cell is:

$$\boldsymbol{M}_{i,j} = \max(\boldsymbol{M}_{i,j-1} - p, \boldsymbol{M}_{i-1,j} - p, \boldsymbol{M}_{i-1,j-1}) + \sigma_{i,j} - c$$

where p is a penalty for insertions and deletions, and c is a bias constant, chosen so that matching segments generate increasing values along the alignment path, and non-matching segments quickly decrease to zero.

The computation of ***M*** proceeds as shown by the shaded area in Figure 2. This evaluation order is intended to find locally similar segments and follow their alignment path. The reason for computing in narrow diagonal bands is that if ***M*** were computed entire row by entire row, all paths would converge to the main diagonal where all frames match perfectly. At each iteration, cells are computed along one row to the left and right of the current path, spanning data that represents a couple of seconds of time. Because of the limited width of the path, references will be made to uninitialized cells in ***M***. These cells and their values are ignored in the maximum value computation.

This algorithm can be further refined. The score along an alignment path will be high at the end of the similar segments, after which the score will decrease to zero. Thus, the algorithm will tend to compute alignment paths that are too long. We can improve on the results by interactively trimming a frame from either end of the path as long as the similarity/length quotient increases. This does not always work well because of local maxima. Another heuristic we use is to trim the final part of a path where the slope is substantially off-diagonal, as shown in Figure 3.

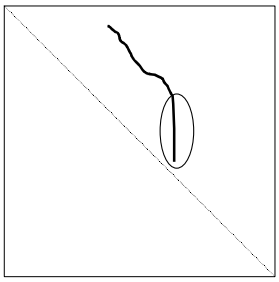

Figure 3. The encircled portion of the alignment path is trimmed because it represents an extreme difference in tempo. The remainder determines a pair of similar segments.

Because the window has a constant size, this algorithm runs in $O(n^2)$ time, and by storing only the portion of the matrix swept by the window, $O(n)$ space. The algorithm is quite efficient in practice.

7. CLUSTERING

After computing pairs of similar segments with any of the three previously described algorithms, we need to form clusters to identify where segments occur more than twice in the music. Essentially, we are just applying the transitivity property of similarity to locate sets of similar segments. However, similarity is not strictly transitive, and the boundaries of segments are imprecise. We must allow approximate matches and somewhat inconsistent data.

We have implemented clustering using a simple algorithm: Start with a set of similar pairs, as computed by algorithms 1, 2, and 3. Remove any pair from the set to form the first cluster. Then search the set for pairs (*a*, *b*) such that either *a* or *b* (or both) is an approximate match to a segment in the cluster. If *a* (or *b*) is not already in the cluster, add it to the cluster. Continue extending the cluster in this way until there are no more similar segments in the set of pairs. Now, repeat this process to form the next cluster, etc., until the set of pairs is empty.

Sometimes, a segment in a cluster will correspond to a subsegment of a pair, e.g. (10, 20) overlaps half of the first segment of the pair ((10, 30), (50, 70)). We do not want to add (10, 30) or (50, 70) to the cluster because these have length 20, whereas the cluster element (10, 20) only has length 10. However, it seems clear that there is a segment similar to (10, 20) starting at 50. In this situation, we split the pair proportionally to synthesize a matching pair. In this case, we would create the pair ((10, 20), (50, 60)) and add (50, 60) to the cluster.

8. ANALYSIS AS EXPLANATION

The final step is to produce an analysis of the musical structure implied by the clusters. We like to view this as an "explanation" process. For each section of music, we "explain" the music in terms of its relationship to other sections. If we could determine relationships of transposition, augmentation, and other forms of variation, these relationships would be part of the explanation. With only similarity, the explanation amounts to labeling music with clusters.

To build an explanation, recall that music is represented by a sequence s_i, $0 \le i < n$. Our goal is to fill in an array E_i, $0 \le i < n$, initially *nil*, with cluster names, indicating which cluster (if any) contains a note or frame of music. The explanation *E* serves to describe the music as a structure based on the repetition and organization of patterns.

Recall that a cluster is a set of intervals. For each *i* in some member of the cluster, we set E_i to the name of the cluster. (Names are arbitrary, e.g. "A", "B", "C", etc.) We then continue searching for the next *i* such that $E_i = nil$ and *i* is in some new cluster. We then label additional points in E_i with this new cluster. However, once a label is set, we do not replace it. This gives priority to musical material that is introduced the earliest, which seems to be a reasonable heuristic to resolve conflicts when clusters overlap.

9. EXAMPLES

9.1 Transcription and Algorithm 1

Figure 4 illustrates an analysis of "Naima" using monophonic transcription and Algorithm 1 to find similar segments. Clusters are shown as heavy lines, which show the location of segments, connected by thin lines. The analysis is shown at the bottom of the figure. The simple "textbook" analysis of this piece would be a presentation of the theme with structure AABA, followed by a piano solo. The saxophone returns to play BA followed by a short coda. In the computer analysis, further structure is discovered within the B part (the bridge), so the computer analysis might be written as AABBCA, where BBC forms the bridge.

The transcription failed to detect more than a few notes of the piano solo. There are a few spurious matching segments here. After the solo, the analysis shows a repetition of the bridge and the A part: BBCA. This is followed by the coda in which there is some repetition. Aside from the solo section, the computer analysis corresponds quite closely to the "textbook" analysis. It can be seen that the A section is half the duration of the B part, which is atypical for an AABA song form. If the program had additional knowledge of standard forms, it might easily guess that this is a slow ballad and uncover additional structure such as the tempo, number of measures, etc. Note, for example, that once the piece is subdivided into segments, further subdivisions are apparent in the RMS amplitude of the audio signal, indicating a duple meter. Additional examples are presented in another paper. [7]

9.2 Chroma and Algorithm 2

Figure 5 illustrates an analysis of Beethoven's "Minuet in G" (performed on piano) using the chroma representation and Algorithm 2 for finding similar segments. Because the repetitions are literal and the composition does not involve improvisation,

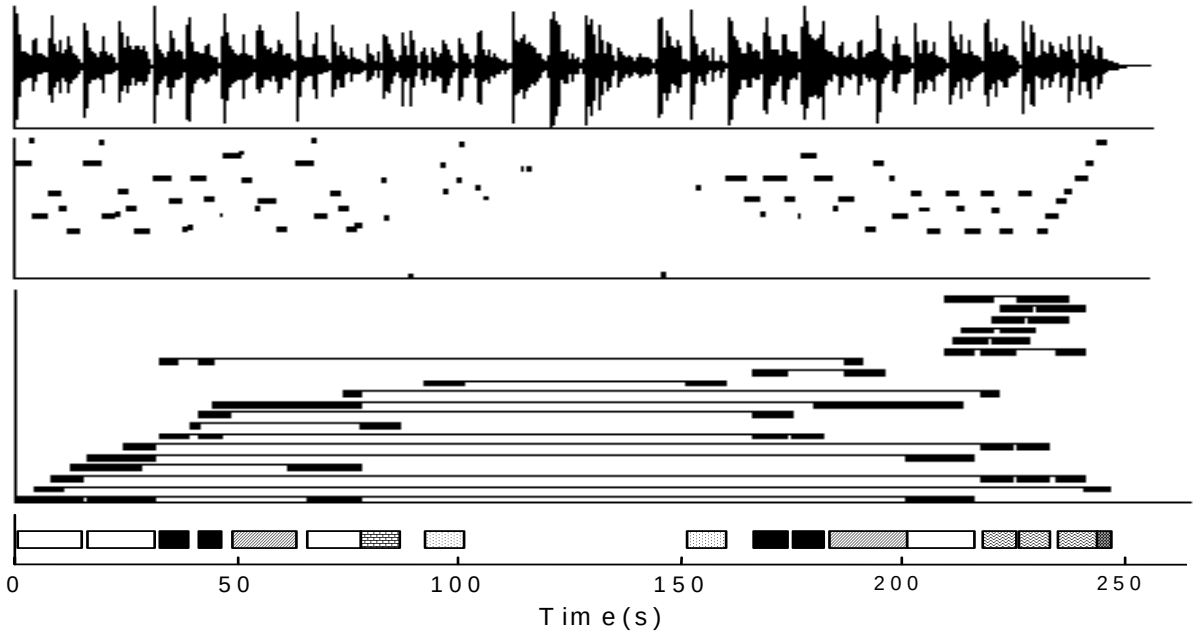

Figure 4. Analysis of Naima. Audio is shown at top. Below that is a transcription shown in piano roll notation. Next is a diagram of clusters. At bottom is the analysis; similar segments are shaded in the same pattern. The form is AABA, where the B part has additional structure that appears as two solid black rectangles and one filled with a "///" pattern. The middle section is a piano solo. The saxophone reenters at the B section, repeats the A part, and ends with a coda consisting of another repetition.

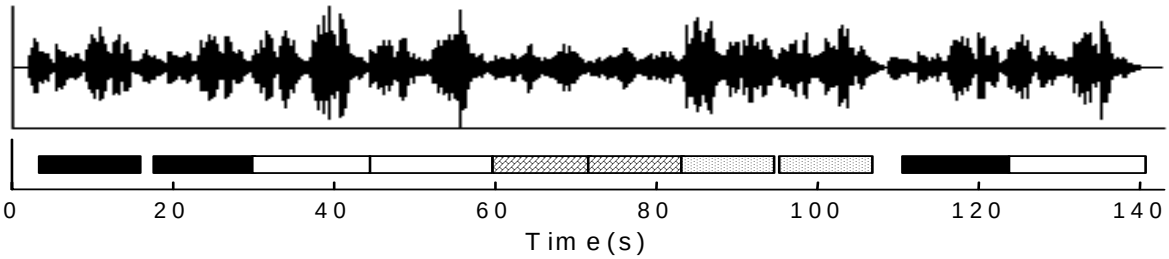

Figure 5. Analysis of Beethoven's Minuet in G performed on piano. The structure, shown at the bottom, is clearly AABBCCDDAB.

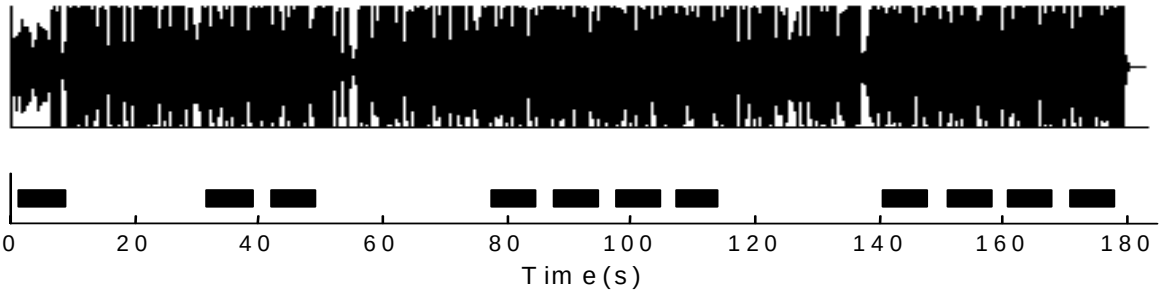

Figure 6. Analysis of a pop song, showing many repetitions of a single segment. Similar segments exist around 20s and 60s, but this similarity was not detected.

the analysis is definitive, revealing that the structure is: AABBCCDDAB.

Figure 6 applies the same techniques to a pop song [17] with considerable repetition. Not all of the song structure was recovered because the repetitions are only approximate; however, the analysis shows a structure that is clearly different from the earlier pieces by Coltrane and Beethoven.

9.3 Polyphonic Transcription & Algorithm 3

So far, polyphonic transcription has not yielded good results as anticipated. Recall that we first transcribe a piece of music and

then construct a harmonic analysis, so the final representation is a sequence of frames, where each frame is a chord. When we listen to the transcriptions, we can hear the original notes and harmony clearly even though many errors are apparent. Similarly, the harmonic analysis of the transcription seems to retain the harmonic structure of the original music. However, the resulting representation does not seem to have clear patterns that are detectable using Algorithm 3. On the other hand, using synthetic data, Algorithm 3 successfully finds matching segments.

The observed problems are probably due to many factors. The analysis often reports different chords when the music is similar; for example, an A minor chord in one segment and C major in the other. Since these chords have 2 pitch classes in common and 2 that are different, σ(Amin, Cmaj) = 0, whereas σ(Cmaj, Cmaj) = 3. Perhaps there is a better similarity function that gives less penalty for plausible chord substitutions. In addition, chord progressions in tonal music tend to use common tones and are based on the 7-note diatonic scale. This tends to make any two chords chosen at random from a given piece of music more similar, leading to false positives. Sometimes Algorithm 3 identifies two segments that have the same single chord, even though the segments are not otherwise similar. A better similarity metric that requires more context might help here. Also, there is a fine line between similar and dissimilar segments, so finding a good value for the bias constant *c* is difficult. Finally, the harmonic analysis may be removing useful information along with the "noise."

To get a better idea of the information content of this representation, Figure 7 is based on an analysis of "Let it Be" performed by the Beatles [13], using polyphonic analysis and chord labeling. After a piano introduction, the vocal melody starts at about 13.5s and finishes the first 4 measures at about 27s. This phrase is repeated throughout the song, so it is interesting to match this known segment against the entire song. Starting at every possible offset, we can search for the best alignment with the score and plot the distance (negative similarity). The distance is zero at 13.5s because the segment matches itself perfectly. The segment repeats almost exactly at about 27s, which appears as a downward spike at 27s. From the graph, it is apparent that the segment also appears with the repetition several other times, as indicated by pairs of downward spikes in the figure.

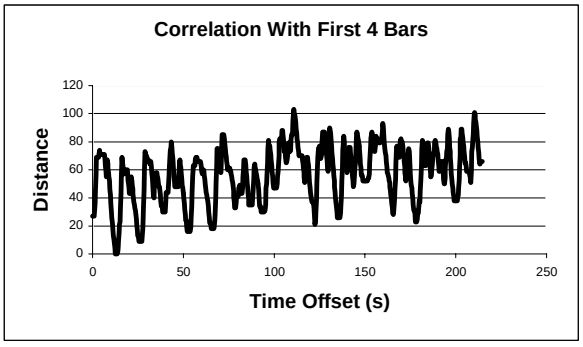

Figure 7. The segment from 13.5s to 27s is aligned at every pointing the score and the distance is plotted. Downward spikes indicate a similar segments, of which there are several.

Figure 7 gives a clear indication that the representation contains information and in fact is finding structure within the music; otherwise, the figure would appear random. Further work is required to use this information to reliably detect similar segements.

10. SUMMARY AND CONCLUSIONS

Music audio presents very difficult problems for music analysis and processing because it contains virtually no structure that is immediately accessible to computers. Unless we solve the complete problem of auditory perception and human intelligence, we must consider more focused efforts to derive structure from audio. In this work, we constructed programs that "listen" to music, recognize repeated patterns, and explain the music in terms of these patterns.

Several techniques can be used to derive a music representation that allows similarity comparison. Monophonic transcription works well if the music consists primarily of one monophonic instrument. Chroma is a simplification of the spectrum and applies to polyphonic material. Polyphonic transcription simplified by harmonic analysis offers another, higher-level representation. Three algorithms for efficiently searching for similar patterns were presented. One of these works with note-level representations from monophonic transcriptions and two work with frame-based representations. We demonstrate through examples that the monophonic and chroma analysis techniques recover a significant, and in some cases, essentially complete top-level structure from audio input.

We find it encouraging that these techniques apply to a range of music, including jazz, classical, and popular recordings. Of course, not all music will work as well as our examples. In particular, through-composed music that develops and transforms musical material rather than simply repeating it cannot be analyzed with our systems. This includes improvised jazz and rock soloing, many vocal styles, and most art music. In spite of these difficulties, we believe the premise that listening is based on recognition of repetition and transformation is still valid. The challenge is to recognize repetition and transformation even when it is not so obvious.

Several areas remain for future work. We are working to better understand the polyphonic transcription data and harmonic analysis, which offer great promise for finding similarity in the face of musical variations. It would be nice to have a formal model that could help to resolve structural ambiguities. For example, a model could enable us to search for patterns and clusters that give the "best" global explanation for observed similarities. Another enhancement to our work would be the use of hierarchy in explanations. This would, for example, support a two-level explanation of the bridge in "Naima." It would be interesting to combine data from beat tracking, key analysis, and other techniques to obtain a more accurate view of music structure. Finally, it would be interesting to find relationships other than repetition. Transposition of small phrases is a common relationship within melodies, but we do not presently detect anything other than repetition. Transposition often occurs in very short sequences, so a good model of musical sequence comparison that incorporates rhythm, harmony, and pitch seems to be necessary to separate random matches from intentional ones.

In conclusion, we offer a set of new techniques and our experience using them to analyze music audio, obtaining structural descriptions. These descriptions are based entirely on the music and its internal structure of similar patterns. Our results suggest this approach is promising for a variety of music processing tasks, including music search, where programs must derive high-level structures and features directly from audio representations.

11. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under award number 0085945. Ann Lewis assisted in the preparation and processing of data. Matija Marolt offered the use of his SONIC transcription software, which enabled us to explore the use of polyphonic transcription for music analysis. We would also like to thank Bryan Pardo for his Harman program and assistance using it. Finally, we thank our other colleagues at the University of Michigan for their collaboration and many stimulating conversations.

12. REFERENCES

[1] Birmingham, W.P., Dannenberg, R.B., Wakefield, G.H., Bartsch, M., Bykowski, D., Mazzoni, D., Meek, C., Mellody, M. and Rand, W., MUSART: Music Retrieval Via Aural Queries. in *International Symposium on Music Information Retrieval*, (Bloomington, Indiana, 2001), 73-81.

[2] Coltrane, J. Naima *Giant Steps*, Atlantic Records, 1960.

[3] Conklin, D. and Anagnostopoulou, C., Representation and Discovery of Multiple Viewpoint Patterns. in *Proceedings of the 2001 International Computer Music Conference*, (2001), International Computer Music Association, 479-485.

[4] Cope, D. *Experiments in Musical Intelligence*. A-R Editions, Inc., Madison, Wisconsin, 1996.

[5] Dannenberg, R.B. Listening to "Naima": An Automated Structural Analysis from Recorded Audio, 2002, (in review).

[6] Dannenberg, R.B., An On-Line Algorithm for Real-Time Accompaniment. in *Proceedings of the 1984 International Computer Music Conference*, (Paris, 1984), International Computer Music Association, 193-198. http://www.cs.cmu.edu/~rbd/bib-accomp.html#icmc84.

[7] Dannenberg, R.B. and Hu, N. Discovering Musical Structure in Audio Recordings, 2002, (in review).

[8] Foote, J. and Cooper, M., Visualizing Musical Structure and Rhythm via Self-Similarity. in *Proceedings of the 2001 International Computer Music Conference*, (Havana, Cuba, 2001), International Computer Music Association, 419-422.

[9] Hewlett, W. and Selfridge-Field, E. (eds.). *Melodic Similarity: Concepts, Procedures, and Applications.* MIT Press, Cambridge, 1998.

[10] Hu, N. and Dannenberg, R.B., A Comparison of Melodic Database Retrieval Techniques Using Sung Queries. in *Joint Conference on Digital Libraries*, (2002), Association for Computing Machinery.

[11] Lartillot, O., Dubnov, S., Assayag, G. and Bejerano, G., Automatic Modeling of Musical Style. in *Proceedings of the 2001 International Computer Music Conference*, (2001), International Computer Music Association, 447-454.

[12] Marolt, M., SONIC: Transcription of Polyphonic Piano Music With Neural Networks. in *Workshop on Current Research Directions in Computer Music*, (Barcelona, 2001), Audiovisual Institute, Pompeu Fabra University, 217-224.

[13] McCartney, P. Let It Be *Let It Be*, Apple Records, 1970.

[14] Meek, C. and Birmingham, W.P., Thematic Extractor. in *2nd Annual International Symposium on Music Information Retrieval*, (Bloomington, Indiana, 2001), Indiana University, 119-128.

[15] Mongeau, M. and Sankoff, D. Comparison of Musical Sequences. in Hewlett, W. and Selfridge-Field, E. eds. *Melodic Similarity Concepts, Procedures, and Applications*, MIT Press, Cambridge, 1990.

[16] Mont-Reynaud, B. and Goldstein, M., On Finding Rhythmic Patterns in Musical Lines. in *Proceedings of the International Computer Music Conference 1985*, (Vancouver, 1985), International Computer Music Association, 391-397.

[17] Mumba, S. Baby Come On Over *Baby Come On Over (CD Single)*, Polydor, 2001.

[18] Narmour, E. Music Expectation by Cognitive Rule-Mapping. *Music Perception*, *17* (3). 329-398.

[19] Pardo, B. Algorithms for Chordal Analysis. *Computer Music Journal*, *26* (2). (in press).

[20] Roads, C. Autocorrelation Pitch Detection. in *The Computer Music Tutorial*, MIT Press, 1996, 509-511.

[21] Rolland, P.-Y. and Ganascia, J.-G. Musical pattern extraction and similarity assessment. in Miranda, E. ed. *Readings in Music and Artificial Intelligence*, Harwood Academic Publishers, 2000, 115-144.

[22] Sankoff, D. and Kruskal, J.B. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

[23] Simon, H.A. and Sumner, R.K. Pattern in Music. in Kleinmuntz, B. ed. *Formal Representation of Human Judgment*, Wiley, New York, 1968.

[24] Stammen, D. and Pennycook, B., Real-Time Recognition of Melodic Fragments Using the Dynamic Timewarp Algorithm. in *Proceedings of the 1993 International Computer Music Conference*, (Tokyo, 1993), International Computer Music Association, 232-235.

---

[1] To understand why, assume there are similar segments, $((i, k), (j, l))$, that overlap, i.e. $0 \le i < j \le k < l < n$. Then, there is some subsegment of (i, k) we will call (i, m) , $m < k$, corresponding to the overlapping region (j, k) and some subsegment of (k, l) we will call (p, l), $p > j$, corresponding to (j, k). Thus, there are three similar segments (i, m), (j, k), and (p, l) that provide an alternate structure to the original overlapping pair. In general, a shorter, more frequent pattern is preferable, so we do not search for overlapping patterns.

[2] An implementation note: for each pair of similar segments, the starting points are implied by the coordinates i, j, but we need to store durations. Since we only search half of the matrix due to symmetry, we store one duration at location i, j and the other at j, i.