Detecting Motives and Recurring Patterns in Polyphonic Music

Paul E. Utgoff (Computer Science)
Chris Raphael (Mathematics & Statistics)
Joshua Stoddard (Mathematics & Statistics)

University of Massachusetts
Amherst, MA 01003 U.S.A.

Technical Report 04-31

May 12, 2004

Abstract

This report considers the problem of detecting and identifying recurring note patterns in polyphonic music. The issue of what it means for a note sequence to be an instance of a pattern is discussed. Following this, the MARPLE algorithm for find finding such patterns is presented. An application of the algorithm illustrates that it finds patterns of interest with a relatively small expenditure of computing resources.

1 Introduction

There is considerable structure in Western classical music. To the extent that structure can be identified and characterized, one has dimensions in which to place and group the various pieces for organizational and analytical purposes. We focus on the problem of identifying recurring note patterns in polyphonic music. Patterns of notes are created deliberately by the composer for the benefit of the listener. Recognizing these patterns is a critical aspect of experiencing and recalling a piece of music.

2 Motives and Recurring Note Patterns

The sounding of a note exists in time and frequency. For analytical purposes, it can be convenient to consider these two dimensions spatially, giving a view of the auditory scene. However, it is critical to keep in mind that music is experienced linearly in time, and the composer writes for this mode of experience. Consider a piano roll. One can listen to it (indirectly), or inspect it visually. Whether it pleases the eye is irrelevant, but the ability to survey the notescape is often convenient for the analyst.

A composer takes many factors into account when writing music, collectively intended to achieve certain effects in the listener. We shall not attempt to recount these effects, nor methods for achieving them, but shall instead focus on just one. Listeners like to gain familiarity with the note figures, even within a single listening, so that there is pleasure both in anticipating their return, and in experiencing them fully when they do. It is tacitly assumed that a composer will present recognizable chunks, and develop them in myriad ways, balancing variety with repetition.

Indeed, both listening satisfaction and analytical insight depend on the ability to hear and recognize note configurations or variants that occur multiple times. This is our concern here, how to find the motives and other recurring note patterns in the music. However, what shall we say constitutes a pattern, or a pattern that was likely intended by the composer? At the very least, a pattern consists of a note configuration that occurs more than once. A short sequence, say of two successive notes differing by a whole step, is likely to occur often by chance (in classical Western music). Nevertheless, we would naturally doubt whether a composer thought of this as a distinctive figure worthy of repetition and development. Longer note sequences that occur often are more likely to be a product of design than of chance.

We have hinted that patterns of note sequences (horizontally related notes) will be more distinctive than patterns of note chords (vertically related notes). This is related to the fact that two chords of identical harmony and identical inversion are often difficult to distinguish. Whether one of the inner voice pitches sounds at a different octave does not change the chord in any distinctive manner. A composer is not likely to construct an easily identifiable pattern by varying inner voices within a single harmony. Of course there can be patterns of harmonies that make distinctive progressions, but our concern here is with patterns of notes. It is the horizontal note patterns that are most distinctive, and we focus on these.

3 The Problem

For convenience, we confine ourselves to the MIDI representation of a score. Each pitch is specified by an integer index into the compass of the standard 12 tones and their octaves, with the value 60 corresponding to middle C. Each note has a specified onset time and offset time. Other performance features such as instrument and key velocity are not of use for our purposes here.

How can we find the recurring patterns within an acceptable allotment of computing effort? An entirely brute-force approach will not work. For example, one cannot enumerate all the possible subsets of notes of a piece, and then collect and count them. For a piece of n notes, there will be 2^n subsets of notes. For a piece of even moderate length, say 2,500 note events, it would be infeasible either to generate or store the 2^{2500} note subsets. Were we to suppose that subsets of size at most 16 notes were of interest, there would be approximately $\sum_{n=2}^{16} \binom{2500}{n}$ subsets, which is still presently intractable. It is necessary to use knowledge of compositional principles to guide the search productively.

In passing, we mention two grammar-based approaches that we will not use, but that naturally come to mind. First, in text and other string languages, grammar induction algorithms and text compression algorithms repeatedly replace commonly occurring substrings with a unique symbol, storing the (symbol,substring) pair as a grammar rule. A shorter representation, as measured by the number of bits to encode the final string and the grammar rules, is deemed better than a longer one because compression is achieved by exploiting regularity (Rissanen & Langdon, 1979).

Notes are not one dimensional, except in the monophonic case (Smith & Medina, 2001; Meek & Birmingham, 2001), so reductions in one dimension will not be applicable. One could imagine discretizing time, and then viewing the time/frequency panorama as a 2-D matrix of Boolean values. This would be much like using graph paper for a piano roll. One could proceed to search for symbol strings in both dimensions simultaneously. However, one cannot generally do a string, or cell-set substitution, because a reduction step cannot be allowed to destroy the data type. When starting with a matrix, one can remove a row or a column, and still be left with a matrix, but removing elements that would not leave a matrix would be unacceptable.

One could instead meld a 2-D group into a unit (without substitution), everywhere it occurs. Repeating this process would allow construction of grammar rules as before. The main problem however is that a search for patterns for grammar rules is nevertheless a search for patterns. An important heuristic emerges however, which is to look for patterns among adjacent and otherwise connected groups of notes. This makes sense for music, in which the notes of a pattern need to be proximal in order to be perceived.

The second grammar-based approach is to employ a graph grammar. Let each note event be represented as a vertex in the graph for the piece. However, where are the edges? Note events are depicted in a music score, but note connections are not typically explicit. Beaming of notes with flags (typically eighths and shorter) is a hint, especially in bygone days, but the composer does not endeavor to identify the patterns. The idea of grouping notes is left to the mind's ear. Connecting every pair of vertices with an edge creates a large graph that must be searched for recurring subgraphs, which is intractable in general. The graph formalism offers no benefit here.

The problem we have set for ourselves is:

Given:

1. MIDI representation of a polyphonic piece of music,
2. Music drawn from the classical Western tradition.

Find:

1. A practical and precise algorithm for detecting and identifying the motives and

recurring note patterns in a piece of music that were likely intended by the composer.

The algorithm should run in a matter of minutes for modest-sized pieces, e.g several thousand notes. It should identify note patterns with no errors of omission (not detecting a pattern that an analyst would) and few errors of commission (alleging a pattern that an analyst would dismiss). For our purpose here, we will not pit algorithm against analyst, but will instead comment informally on the output of the algorithm. At this point, our research endeavor is largely exploratory, to determine the feasibility of such an approach.

4 The MARPLE Algorithm

The MARPLE algorithm (Motives And Recurring Patterns LExicon) is intended to meet the specifications described above. It will be necessary to bring musical knowledge to bear in order to render the search for patterns feasible. Each such knowledge element that is harnessed to guide the search constitutes a heuristic, and each is described here as such. Our notion of heuristic is broad, referring to any aspect of a search procedure that accelerates the search through the virtual space of all possible note patterns within a piece of music. Our description of the MARPLE algorithm proceeds by describing the heuristics that ultimately comprise the search procedure.

The **first heuristic** is that the search will consider first those sequences of length $k = 2$, followed by the sequences of length $k = 3$, through increasing k until a natural limit has been reached. This is based on the knowledge that composers create patterns that are long enough to be distinctive yet short enough to be retained quickly and reliably. Should an entire section that is repeated once literally count as a pattern? We say no, because it is too long to retain and because it occurs too infrequently. Similarly, we take the liberty of ruling out a simple scale step, even though it is simple and occurs often, because it is not distinctive.

The **second heuristic** is that patterns consist of consecutive notes, possibly articulated with non-disruptive rests. This is based on the notion of sequential coherence, which was mentioned above. A sequence that contains too large a break is not one sequence, but instead two or more. We define a predicate $\mathit{consecutive}(n_1, n_2)$ of two notes that is true if and only the pair of notes can be considered to be consecutive. Two notes can be considered as consecutive if they are within an octave of each other, chronologically in order, and if the length of the silence or overlap between the two notes is small with respect to the duration of each of the two notes. Let $\mathit{on}(n)$ indicate the onset time for note n, $\mathit{off}(n)$ indicate the offset time for note n, and $\mathit{pitch}(n)$ denote the MIDI pitch for note n. The default value of the δ parameter below is 0.4. Define:

$$\begin{aligned}
\mathit{consecutive}(n_1,n_2) \quad \leftrightarrow \quad & n_1 \neq n_2 \wedge \\
& \mathit{on}(n_1) \leq \mathit{on}(n_2) \wedge \\
& |\mathit{pitch}(n_1) - \mathit{pitch}(n_2)| \leq 12 \wedge \\
& \mathit{on}(n_2) - \mathit{off}(n_1) \geq 0 \to \left(\frac{on(n_2) - off(n_1)}{on(n_2) - on(n_1)} < \delta \wedge \frac{on(n_2) - off(n_1)}{off(n_2) - off(n_1)} < \delta\right) \wedge \\
& \mathit{on}(n_2) - \mathit{off}(n_1) < 0 \to \left(\frac{off(n_1) - on(n_2)}{off(n_1) - on(n_1)} < \delta \wedge \frac{off(n_1) - on(n_2)}{off(n_2) - on(n_2)} < \delta\right) \qquad (1)
\end{aligned}$$

Define the set of all 2-graphs (digraphs) D to be the set of all possible note pairs (n_i, n_j) that satisfy the predicate $\mathit{consecutive}(n_i, n_j)$. This set is important because it defines all possible

1. Form the set of 2-graphs from all possible pairs of consecutive notes. Set k to 2.
2. Increment k. Generate all k-graphs from which the last note of a $(k-1)$-graph and the first note of a 2-graph are consecutive.
3. Form the clusters that group the new instance sequences according to similarity.
4. Sort the clusters by size, and eliminate those smaller than β (default 8).
5. if $k \geq 4$
   (a) Compare every sequence s_i of length k to every sequence s_j of length $k-1$. If s_j is a prefix or a suffix of s_i, then eliminate s_j.
   (b) Recluster the sequences of length $k-1$, and eliminate the clusters of length $k-1$ of size less than β.
6. If clusters of length k remain, go to Step 2.

Table 1. The MARPLE Algorithm

consecutive note pairs. The set of consecutive note sequences of greater length can be composed from these pairs, and no other note sequences need be considered. In general:

$$(\forall i,j,k) consecutive(n_i,n_j) \wedge consecutive(n_j,n_k) \rightarrow consecutive(n_i,n_j,n_k)$$

This method of composing 2-graphs to produce higher order k-graphs is fundamental to the MARPLE algorithm.

Although we have said which note sequences can be considered to be consecutive, we have defined only instances of possible patterns. How is a pattern detected? A note sequence that is highly similar to another hints at a pattern. To the extent that there are many similar instances, we can assert the presence of a pattern. To this end, the MARPLE algorithm forms clusters of similar observed sequences.

The clustering method is simply to find an instance that does not belong to a cluster, define a new cluster with that instance as its prototype, and then to absorb all clusterless instances that are sufficiently similar to the prototype. This is repeated until every instance belongs to a cluster. A cluster with a sufficient number of members represents a pattern. Cluster size is the count of the instances that belong to the cluster. As mentioned above, a pattern must be distinctive, not just abundant. To the extent that a pattern is long (the cluster's instances are long), it is distinctive. A pattern can also be defined negatively, as a prototypical sequence that has not too few exemplars and is not too short in length.

Two note sequences can be compared with respect to their corresponding pairwise time steps and pitch steps. From a score, similarly a transcribed MIDI file, the indicated onset times are given precisely. A MIDI file from a human performance would require registering with a score, but this is beyond the scope of the present discussion. Alternatively, one could accept some variation in the time step lengths. Assuming precise onset and offset, one can compare the time-step lengths precisely. If the length of the time steps in one sequence is a scalar multiple of the time steps of the

Length	Cluster Sizes
3	126 72 64 32 27 24 20 18 17 17 15 14 14 13 13 12 12 12 12 11 11 11 11 10 10 10 9 9 9 9 8 8 8 8 8 8 8 8 8 8 8
4	49 45 28 23 22 18 17 17 16 15 14 14 13 12 12 11 11 10 9 9 9 9 9 8 8 8 8 8 8 8
5	10 8
6	11 9 9 8 8
7	35 8
8	19 14 10 8 8 8

Table 2. Cluster Sizes

other sequence, then rhythmic augmentation or diminution has been detected and can be factored out (normalized).

The stepwise comparison of frequency steps requires considerable flexibility. For example, a note sequence expressed in a major modality, and the otherwise same sequence appearing elsewhere in a minor modality, will have numerous stepwise pitch differences. Typically these will be just one or two half-steps. We shall see specific examples of tolerable differences below. The similarity metric is defined here as a dissimilarity or distance metric. For brevity, we define a term $(x \rightarrow v)$ to have value v if condition x is true, and to have value 0 otherwise. The distance metric can be expressed as a sum of such terms and other expressions.

Let $df_{1,i}$ be the pitch difference between notes i and $i+1$ in sequence s_1, and let $df_{2,i}$ be the pitch difference between notes i and $i+1$ in sequence s_2. Define Let dp_i be $df_{1,i} - df_{2,i}$, which is the difference in the size of two corresponding pitch steps. Define:

$$\begin{aligned} \mathit{distance}(s_1,s_2) \;=\; & (\mathit{common\ notes} \rightarrow 100) + \\ & (\mathit{contrary\ motion} \rightarrow 100) + \\ & (\mathit{unequal\ time\ steps} \rightarrow 100) + \\ & \sum_i (|dp_i| > 3 \rightarrow |dp_i| - 2) \end{aligned} \tag{2}$$

The **third heuristic** is to eliminate clusters of small size. No search for $(k+1)$-graphs can produce a larger cluster with the same prefix, so there is no utility in retaining the clusters that are already deemed too small to constitute a pattern of interest. The minimum size for a cluster to survive elimination is specified by a parameter β (default value 8) described below.

It is essential to remove patterns that are subsumed by others (Lartillot, 2003). Suppose that there is a pattern of 8-notes to be found. No less frequent will be the shorter sequences contained within it. There will be two patterns of length 7, three of length 6, and generally $h+1$ patterns of length $m-h$, where m is the length of the pattern, and $m-h$ is the length of the sub-pattern, with $h \in [1, m-2]$.

An efficient procedure for eliminating sub-patterns is described below with the algorithm, and one can view this as a **fourth heuristic**. It follows from the fact that if one has in hand the clusters of length k, then one can eliminate exactly those sequences of length $k-1$ that are subsumed by a sequence of length k that is a member of a sufficiently large cluster. Only two alignments need to

Meas	Pos	Contour
53	1	72 - 2 - 1 + 1 - 1 - 2 + 2
54	1	79 - 2 - 1 + 3 - 2 - 1 + 1
55	1	69 - 2 - 2 + 4 - 2 - 2 + 2
56	1	77 - 1 - 2 + 3 - 1 - 2 + 2
57	1	67 - 2 - 1 + 3 - 2 - 1 + 1
58	1	76 - 2 - 2 + 4 - 2 - 2 + 2
59	1	55 - 2 - 1 + 1 - 1 - 2 + 2
60	1	53 - 1 - 2 + 2 - 2 - 2 + 2
61	1	52 - 2 - 2 + 2 - 2 - 1 + 1
62	1	50 - 2 - 1 + 1 - 1 - 2 + 2
63	1	48 - 1 - 2 + 2 - 2 - 2 + 2
104	1	69 - 2 - 2 + 2 - 2 - 1 + 1
105	1	74 - 2 - 2 + 2 - 2 - 1 + 3
106	1	67 - 2 - 1 + 1 - 1 - 2 + 3
107	1	79 - 2 - 1 + 1 - 1 - 2 + 3
108	1	48 - 2 - 1 + 1 - 1 - 2 + 3
109	1	84 - 2 - 1 + 1 - 1 - 2 + 3
110	1	65 - 2 - 1 + 1 - 1 - 2 + 3
111	1	77 - 2 - 1 + 1 - 1 - 2 + 3
112	1	70 - 1 - 2 + 2 - 2 - 1 + 3
117	1	74 - 2 - 2 + 2 - 2 - 1 + 3
118	1	55 - 2 - 1 + 1 - 1 - 2 + 3
119	1	77 - 1 - 2 + 2 - 2 - 1 + 1
120	1	67 - 2 - 1 + 1 - 1 - 2 + 3
121	1	69 - 2 - 2 + 2 - 2 - 1 + 3
175	1	65 - 2 - 1 + 1 - 1 - 2 + 2
177	1	62 - 2 - 2 + 4 - 2 - 2 + 2
178	1	70 - 1 - 2 + 3 - 1 - 2 + 2
179	1	60 - 2 - 1 + 3 - 2 - 1 + 1
180	1	69 - 2 - 2 + 4 - 2 - 2 + 2
181	1	48 - 2 - 1 + 1 - 1 - 2 + 2
182	1	46 - 1 - 2 + 2 - 2 - 2 + 2
183	1	45 - 2 - 2 + 2 - 2 - 1 + 1
184	1	43 - 2 - 1 + 1 - 1 - 2 + 2
185	1	41 - 1 - 2 + 2 - 2 - 2 + 2

Table 3. The Largest Length 7 Cluster

be considered for the two sequences to be compared. After removing the subsumed sequences of length $k-1$, all the sequences of length $k-1$ can be reclustered.

The basic MARPLE algorithm is shown in Table 1. Two refinements remain. One is achieved by modification of the definition of *consecutive* and the other by adding a term to the *distance* function.

The **fifth heuristic** is to consider any pair of consecutive notes that straddle a measure boundary, tied or not, to be not consecutive. This comes from experience with an earlier version of the algorithm. Some patterns that straddle measure boundaries can cause counterproductive elimina-

Meas	Pos	Contour
77	0	67 - 3 - 4 + 4 + 3 - 3 + 6 - 6
78	0	69 - 4 - 5 + 5 + 4 - 4 + 7 - 7
79	0	67 - 3 - 4 + 4 + 3 - 3 + 6 - 3
80	0	69 - 4 - 5 + 5 + 4 - 4 + 5 - 5
81	0	72 - 3 - 4 + 4 + 3 - 3 + 6 - 6
82	0	74 - 4 - 5 + 5 + 4 - 4 + 7 - 7
83	0	72 - 3 - 4 + 4 + 3 - 3 + 6 - 6
84	0	74 - 4 - 5 + 5 + 4 - 4 + 7 - 7
85	0	79 - 6 - 3 + 3 + 6 - 6 + 6 - 5
87	0	79 - 6 - 3 + 3 + 6 - 6 + 6 - 5
89	0	79 - 6 - 3 + 3 + 6 - 6 + 6 - 5
156	0	74 - 4 - 5 + 5 + 4 - 4 + 7 - 7
157	0	72 - 3 - 4 + 4 + 3 - 3 + 6 - 6
160	0	76 - 4 - 5 + 5 + 4 - 4 + 7 - 7
161	0	74 - 3 - 4 + 4 + 3 - 3 + 6 - 6
162	0	76 - 4 - 5 + 5 + 4 - 4 + 5 - 5
163	0	79 - 3 - 4 + 4 + 3 - 3 + 6 - 6
164	0	81 - 4 - 5 + 5 + 4 - 4 + 7 - 7
165	0	79 - 3 - 4 + 4 + 3 - 3 + 6 - 6

Table 4. The Largest Length 8 Cluster

tions of subsequences. This constraint can be seen as an addendum to the second heuristic, and it is implemented within the definition of *consecutive* by adding the three additional conjunctive terms $\mathit{straddle}(n_1)$, $\mathit{straddle}(n_2)$ and $\mathit{position}(n_1) < \mathit{position}(n_2)$. The algorithm runs more quickly with this constraint, but it runs efficiently without it. The issue here is the quality of the patterns that are produced.

Finally, the **sixth heuristic** takes advantage of the fact that most instances of a pattern begin at the same location with respect to the beat. This constraint is implemented in the distance function by assigning a large killer value, e.g. 100, to two sequences if they do not commence at the same point in a measure. Indeed, this phase constraint among the instances of a pattern facilitates determining the meter, though this is not done by MARPLE. This constraint is imperfect as it is known that composers sometimes stray deliberately from this practice.

5 Illustration

The MARPLE algorithm was applied to the third movement of J.S. Bach's Italian Concerto, BWV 971c. This composer is known for recombining a relatively small amount of material in varied and often intricate ways, and this piece is replete with patterns. For this reason, it serves as a good test of the algorithm.

The piece is in cut time, with eighth notes as the shortest value, ignoring ornaments. The MARPLE algorithm finds all the patterns of lengths 3 through 8 reasonably quickly (12 seconds on a 0.75G Pentium). Table 2 shows the cluster sizes for each sequence length. Notice how the number and size of the clusters falls with increasing length. Nevertheless, the few clusters contain many exemplars.

It is instructive to examine two of the clusters. All 35 of the instances of the largest cluster

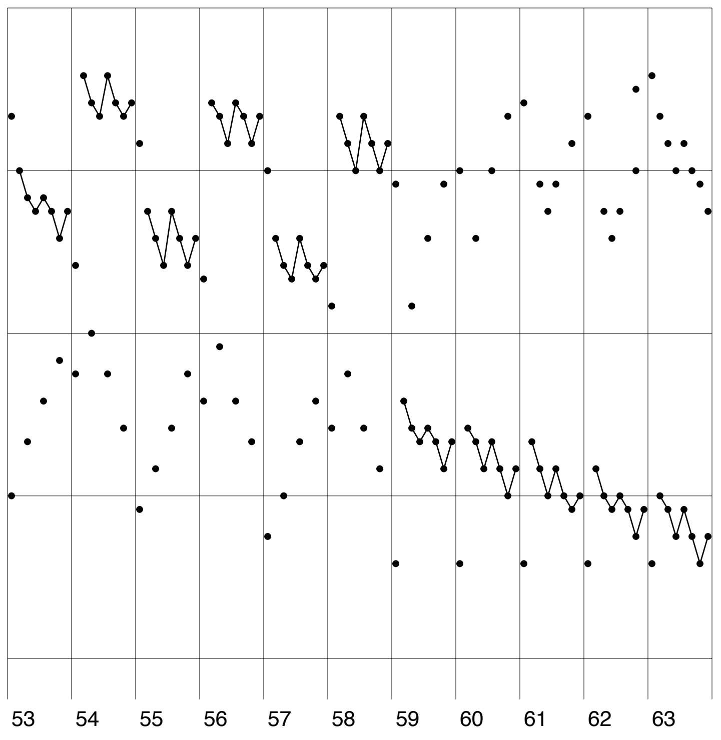

Figure 1. Instances in Measures 53-63; Vertical dimension is MIDI pitch

of length 7 are shown in Table 3. These are all of the cases of this particular pattern in the score. All the notes are eighths, and every sequence starts on the first half beat of the measure (half beat #1, from #0 to #7). The sequences start on a variety of pitches, and the pitch contours vary by as many as three half-steps. As depicted in Figure 1, the pattern appears in eleven measures in a row, starting at measure 53, with no identical contours. This pattern is distinctive to the ear, and its measure position helps it to stand out. The intervals consist of seconds and thirds.

The 19 instances of the largest cluster of length 8 are shown in Table 4. All the notes are eighths, and every sequence starts on the first beat of the measure (half beat #0). The sequences start on a variety of pitches, and the pitch contours vary by as many as three half-steps. The pattern is usually a broken chord, arpeggiated in a distinctive sequence. In some instances of the pattern, one sees some movement of an inner voice. The pattern is distinctive to the ear. The column of measure numbers suggests two blocks of measures for this pattern. One then wonders what happened in measures 86, 88, 158, and 159. For 86 and 88, one interval is too different from the prototype, though an analyst would discount this immediately. For 158 and 159, a tied half note disturbs the pattern, but is a rather obvious dismissible variation.

A second cluster of length 8, not shown, is of interest. If the first eighth of each sequence were removed, the sequence would be an instance of the 7-note pattern discussed above. In this musical context however, Bach finishes the previous figure on the down beat and then proceeds immediately. A performer would provide an articulation here to help the listener notice the 7-note pattern even here. It seems that in this case, the great abundance (35 instances) of the 7-note pattern

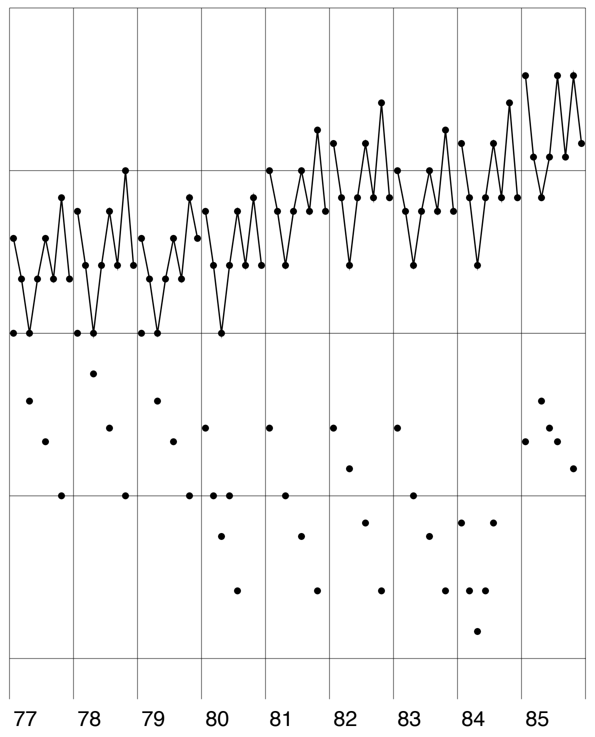

Figure 2. Instances in Measures 77-85; Vertical dimension is MIDI pitch

would cast doubt on the 8 inclusive instances of the 8-note pattern.

A final point is worth mentioning. There are a several locations with sequence of full chords. Each path through the notes of the chords, assuming that the consecutive predicate is satisfied, is a possible sequence. Suppose that there were five chords of three notes each, played consecutively as define above. There would be $3^5 = 243$ possible sequences. The clustering process will reduce this to three possible clusters because sequences that share notes cannot be in the same cluster. Nevertheless, some three clusters would be formed. If the chord sequence occurs sufficiently often, then each of these three clusters will constitute a pattern. Special treatment of simultaneous patterns is needed.

6 Conclusions

We have shown that by bringing musical knowledge to bear, it is possible to construct a practical search procedure for detecting and identifying motives and recurring note patterns in polyphonic music. Work remains to be done, some of which has been outlined above. This will include applying the approach to a larger sample of input files representing different composers and different stylistic periods.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0113496. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The MIDI file was created by A. Hut, and made available through the Classical Music Archive at http://www.classicalarchives.com.

References

Lartillot, O. (2003). Discovering musical patterns through perceptive heuristics. *Proceedings of the Fourth Annual International Symposium on Music Information Retrieval* (pp. 89-96). Baltimore, MD.

Meek, C., & Birmingham, W. P. (2001). Thematic extractor. *Proceedings of the Second Annual International Symposium on Music Information Retrieval* (pp. 119-128). Bloomington, IN.

Rissanen, J., & Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of Research and Development, 23*, 149-162.

Smith, L., & Medina, R. (2001). Discovering themes by exact pattern matching. *Proceedings of the Second Annual International Symposium on Music Information Retrieval* (pp. 31-32). Bloomington, IN.