Detecting Motives and Recurring Patterns in Polyphonic Music*

Paul E. Utgoff and Phillip B. Kirlin
University of Massachusetts
Amherst, MA 01003 U.S.A.
{utgoff|pkirlin}@cs.umass.edu

Abstract

We consider the problem of detecting and identifying recurring note patterns in polyphonic music. A practical algorithm MARPLE for find finding such patterns is presented. The algorithm is evaluated on sequenced MIDI files for the 96 pieces of the two books of Bach's Well-Tempered Clavier. Strengths and weaknesses are identified.

1 Introduction

There is considerable structure in Western classical music. 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, understanding, and recalling a piece of music.

2 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 on a suitable device, 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, which are collectively intended to achieve certain effects for 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 musical entitites, 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.

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 repeatedly are more likely to be a product of design than of chance. A pattern that constitutes a theme differs from a pattern that simply provides texture.

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 due to the fact that two chords of identical harmony and identical inversion are often difficult to distinguish aurally. Whether one of the inner voice pitches sounds at a different octave, without altering the inversion, 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 horizontal patterns of notes. The horizontal note patterns are most distinctive, and we focus on how to find these automatically.

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 twelve tones and their

*Appeared in *Proceedings of the International Computer Music Conference, pp. 487-494*

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. Much information available in a score is not available in the MIDI representation, such as rests and note beaming.

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 $2^{2,500}$ note subsets. Instead, 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 solely sequential, 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 and 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 perform a string, or cell-set substitution, because a reduction step cannot be allowed to destroy the data type, in this case the matrix. 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 as coherent.

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. An arrangement of the music into parts for the performers also gives some indication of grouping. A composer does not otherwise endeavor to indicate the patterns. The idea of grouping notes is left to the mind's ear (Lerdahl & Jackendoff, 1983). 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 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).

4 Related Work

One approach is to segment the notes of a piece into monophonic note sequences, convert the sequences to strings, and then to apply various string matching and clustering techniques. Several investigations have assumed that the segmentation into monophonic sequences is a pre-processing step, to be done by hand or by means of a separate program (Kirlin & Utgoff, 2005). This decomposes the problem, which is often a good idea, but it also eliminates some of the fundamental aspects of polyphonic pattern finding. For example, Buxtehude's fugues often contain interesting voice crossing. One can separate these into monophonic sequences only by solving the original polyphonic pattern finding problem. Human experts seem to find the patterns and the voices simultaneously because one informs the other, but we have not come across any such system designs.

Of course, even with monophonic sequences, a challenging task remains. Cambouropoulos & Widmer (2002) present a system for extracting note patterns from monophonic sequences. They address issues related to unifying core note patterns that may have differing elaborations (Lerdahl & Jackendoff, 1983), which is critical for noticing variations. They make use of edit distance for string matching, which helps to handle variations. Jan (2004) discusses searching for patterns using Huron's Humdrum Toolkit. One defines a pattern by hand, in a string language, and then invokes the tool to search multiple music files for portions that match, based on Unix utilities for matching regular expressions. Lartillot (2004, 2005) discusses issues regarding pattern discovery in monophonic sequences from a piece of music. Rolland (2001) describes searching for patterns from monophonic sequences taken from muliple pieces of music.

Conklin (2002) presents a method for hierarchical representation of music objects. Multiple views can be constructed above the ground objects. One application is to finding patterns in harmonic progressions.

Meredith et al (2002) map each note, in terms of its pitch and onset time to a point in 2-D space. Operationally their program works with numeric vectors, which represent ordered tuples of information. There is analysis of run-time complexity, with empirical validation. The discussion of pattern finding is anecdotal. The approach does not include a method for filtering and otherwise prioritizing the "tens of thousands" of patterns that it generates. Patil & Mundur (2005) discuss an n-gram based approach to finding note patterns in synthetic audio data. No results are given, but they present a useful metric for the interestingness of a pattern. The work we describe below is most closely related to these two approaches to finding note patterns within a polyphonic piece of music.

5 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 element of musical knowledge 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 no more patterns are to be found. 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. A particular note sequence is one monophonic path through the polyphonic notescape. Should an entire section that is repeated once literally count as a pattern? 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. A specific measure of pattern interest is given below.

The **second heuristic** is that patterns consist of consecutive notes, possibly articulated with 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) shorter sequences. We define a predicate $consecutive(n_1, n_2)$ of two notes that is true if and only if the pair of notes can be considered to be consecutive. Two notes can be considered as consecutive if they are not too distant in pitch from each other, chronologically in order by onset time, 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 $on(n)$ indicate the onset time for note n, $off(n)$ indicate the offset time for note n, and $pitch(n)$ denote the MIDI pitch for note n. The default value of the consecutiveness parameter δ below is 0.4, and the default value of the pitch distance parameter γ is 12, corresponding to one octave. Define:

$$
\begin{aligned}
& consecutive(n_1,n_2) \leftrightarrow \\
& \quad n_1 \neq n_2 \wedge \\
& \quad on(n_1) \leq on(n_2) \wedge \\
& \quad |\, pitch(n_1) - pitch(n_2)| \leq \gamma \wedge \\
& \quad on(n_2) - off(n_1) \geq 0 \rightarrow \\
& \qquad \left(\tfrac{on(n_2)-off(n_1)}{on(n_2)-on(n_1)} < \delta \wedge \tfrac{on(n_2)-off(n_1)}{off(n_2)-off(n_1)} < \delta\right) \wedge \\
& \quad on(n_2) - off(n_1) < 0 \rightarrow \\
& \qquad \left(\tfrac{off(n_1)-on(n_2)}{off(n_1)-on(n_1)} < \delta \wedge \tfrac{off(n_1)-on(n_2)}{off(n_2)-on(n_2)} < \delta\right)
\end{aligned}
$$

Define the set of all 2-grams (di-grams, synonymously bigrams) D to be the set of all possible note pairs (n_i, n_j) that satisfy the predicate $consecutive(n_i, n_j)$. This set is important because it defines all possible 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:

$$
\begin{aligned}
& (\forall i,j,k)\ consecutive(n_i,n_j) \wedge consecutive(n_j,n_k) \rightarrow \\
& \quad consecutive(n_i,n_j,n_k)
\end{aligned}
$$

This method of composing 2-grams to produce higher order k-grams 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

Table 1: The MARPLE Algorithm

1. Form the set of 2-grams from all possible pairs of consecutive notes. Set k to 2.
2. Increment k. Generate all k-grams from which the last note of a $(k-1)$-gram and the first note of a 2-gram are the same.
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 5).
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.
7. Rank all of the clusters by the interest metric.

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 simply finds an instance that does not belong to a cluster, defines a new cluster with that instance as its prototype, and then absorbs 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.

Two note sequences can be compared with respect to their corresponding pairwise time steps and pitch steps. From a score, similarly a sequenced MIDI file, the indicated onset times occur at common multiples of the basic unit, without temporal variation. 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 reasonably regular onset and offset multiples, one can compare the time-step lengths meaningfully. If the length of the time steps in one sequence is a scalar multiple of the time steps of the 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. Let dp_i be $df_{1,i} - df_{2,i}$, which is the difference in the size of two corresponding pitch steps. Define:

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

The **third heuristic** is to eliminate clusters of small size. No search for $(k+1)$-grams 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 5) 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 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. One refinement remains; the **fifth heuristic** takes advantage of the fact that most instances of a pattern begin at the same location with respect to the measure boundaries. This constraint is implemented in the distance function by adding a large killer value of 1000 to the measured distance if the two sequences do not commence at the same point in a measure. One can set the size of a measure, and if it differs from that specified by the piece, then we would call it an *artificial measure*. Typically, an artificial measure is shorter than the true measure. As an extreme, one can set the length of the artificial measure to be just one beat of the shortest note or rest duration in the piece, effectively disabling this constraint. This phase constraint among the instances of a pattern facilitates determining the meter, though this is not done by MARPLE. For MARPLE, one must find by hand the shortest MIDI note length for which every longer note length is a multiple, which is the greatest common divisor. Then, by hand, one sets the number of the shortest note lengths that constitute a measure's worth of time.

The final step of the MARPLE algorithm is to sort the clusters by a numeric measure of how interesting the cluster is musically. The 'interest' of a cluster of n-grams is a subjective measure, which we define to be the natural log of the number of n-grams in the cluster times the average interest of the n-grams in the cluster. The interest of an individual n-gram is a function of the variety in the stream of note durations and also of recency of the twelve chromatic pitches. More specifically, for the n notes of an n-gram, there are $n-1$ consecutive pairs of notes. For each pair, add 1 to the duration interest if the two notes of the pair are of differing durations, and add 0 otherwise. Divide the resulting sum by 100, for scaling purposes. The pitch interest is a function of the recency of a chromatic pitch. It is the natural log of the number of unit durations since 1 unit before the start of the n-gram, or since the previous occurrence of that pitch in the same n-gram, whichever is more later. These computations are restated:

$$\begin{aligned} &\mathit{interest}(c) = \\ &\quad \ln(\mathit{size}(c) \cdot \\ &\quad (\textstyle\sum_{i=1}^{n} \ln(\mathit{recent}(\mathit{pitch}_i)) + \\ &\quad \tfrac{1}{100} \cdot \textstyle\sum_{i=1}^{n-1} \mathit{durdif}(\mathit{note}_i, \mathit{note}_{i+1}))) \end{aligned}$$

It is not straightforward to provide a meaningful description of the run-time complexity of the MARPLE algorithm. Run-times are typically specified in terms of input size, which in this case would be the number of notes in the piece of music. However, the running time of our algorithm relates most closely to the size and prevalence of patterns in the music, which may or may not be related to the number of notes. The worst case complexity of MARPLE is too pessimistic to be of any use. Consider an extreme piece of music that consists of n consecutive quarter notes of chords, each chord consisting of the same k consecutive (adjacent) pitches. Such a piece would contain k^n possible paths through it. This is already exponential in n, without having accounted for the $O(n^2)$ subsequences within each path. Music is typically much simpler, but that takes us toward expected run-time complexity. It is not clear to us at the moment what the independent variable(s) should be for such an analysis. Complexity does not seem to be related to the number of notes, but rather to the regularities in their arrangement.

6 An Evaluation of MARPLE

How well does MARPLE work? Making this question precise, and then trying to answer it produces an experiment. We developed MARPLE using a variety of input MIDI files, almost entirely from the Baroque era, and mostly by Bach, including his fifteen inventions. Of course it would be useful to cast a wider net, and this will come in time. How well does MARPLE do on pieces from the Baroque (and before)?

We decided to test MARPLE on both books of Bach's Well-Tempered Clavier (WTC). Each contains 24 preludes and fugues, for a total of 96 individual pieces. None of these pieces had been tested with MARPLE beforehand, nor used during its development. For any given input, MARPLE will typically find many patterns of interest. How often are the patterns that a trained musician would find to be most salient found by MARPLE ranked equally highly? More precisely, and to avoid brittle yes/no measurements, we ask at what rank, by MARPLE, does the most salient pattern identified by a scholar appear? For the WTC, it is not particularly difficult to identify the main 'theme' of each piece. Fugues are particularly easy due to the form. Nevertheless, we consulted Barlow & Morgenstern (1948) as an authority on the themes for these 96 pieces.

Table 2 shows parameter settings and the result for each of the 96 pieces. The shortest note value in each piece is the unit of measure, and the number of such units defines the length of the artificial measure. These are the two piece-specific parameters, set by hand for each piece. The algorithmic parameters were left at their default values, as described above, at all times.

The shortest note value can almost always be determined by inspection. In some cases however, such as the D minor Fugue of Book II, the theme contains triplets. In such a case, one needs to select a smallest unit that works for the triplets as well as the other note values. In this case, a sixteenth note is 96 ticks, and a note of a triplet is 64 ticks. The greatest common divisor is 32, so the unit of measure is 32 ticks, which is one third of a sixteenth note, hence a 48th note. This calculation is done by hand.

The size of the artificial measure could be set so that the

Table 2: Experiment for Well-Tempered Clavier

		Book I Prelude			Book II Prelude			Book I Fugue			Book II Fugue		
No.	Key	unit	meas	rank	unit	meas	rank	unit	meas	rank	unit	meas	rank
1	C maj	𝅘𝅥𝅯	8	s	𝅘𝅥𝅰	16	—	𝅘𝅥𝅰	16	1	𝅘𝅥𝅯	4	4r
2	C min	𝅘𝅥𝅱	32	s	𝅘𝅥𝅯	8	16	𝅘𝅥𝅯	8	1	𝅘𝅥𝅰	16	1
3	C♯ maj	𝅘𝅥𝅯	6	1	𝅘𝅥𝅯	2	—	𝅘𝅥𝅯	8	19	𝅘𝅥𝅰	16	—r
4	C♯ min	𝅘𝅥𝅮	6	8	𝅘𝅥𝅰	12	—	𝅘𝅥𝅮	4	1	𝅘𝅥𝅯	6	1
5	D maj	𝅘𝅥𝅰	16	1	𝅘𝅥𝅯	12	1	𝅘𝅥𝅰	8	6	𝅘𝅥𝅰	8	1
6	D min	𝅘𝅥𝅯	12	4	𝅘𝅥𝅯	4	1	𝅘𝅥𝅯	4	4	𝅘𝅥𝅯/3	12	1
7	E♭ maj	𝅘𝅥𝅰	16	6	𝅘𝅥𝅯	6	7	𝅘𝅥𝅯	8	19	𝅘𝅥𝅮	4	1r
8	E♭ min	𝅘𝅥𝅰	24	s	𝅘𝅥𝅰	16	41	𝅘𝅥𝅯	8	2	𝅘𝅥𝅯	8	1
9	E maj	𝅘𝅥𝅯	12	1	𝅘𝅥𝅰	8	159	𝅘𝅥𝅯	8	1r	𝅘𝅥𝅮	8	2
10	E min	𝅘𝅥𝅰	16	—	𝅘𝅥𝅰	12	—	𝅘𝅥𝅯	4	1	𝅘𝅥𝅯/3	12	1
11	F maj	𝅘𝅥𝅯	12	1	𝅘𝅥𝅮	4	1	𝅘𝅥𝅯	6	1	𝅘𝅥𝅰	6	6r
12	F min	𝅘𝅥𝅰	16	1	𝅘𝅥𝅯	4	—	𝅘𝅥𝅯	8	1	𝅘𝅥𝅯	4	1
13	F♯ maj	𝅘𝅥𝅯	6	9	𝅘𝅥𝅰	8	—,6	𝅘𝅥𝅯	8	3	𝅘𝅥𝅯	8	22r
14	F♯ min	𝅘𝅥𝅯	8	1	𝅘𝅥𝅯/3	12	6	𝅘𝅥𝅯	12	1	𝅘𝅥𝅱	8	6
15	G maj	𝅘𝅥𝅯	12	1	𝅘𝅥𝅯	4	1	𝅘𝅥𝅰	12	33	𝅘𝅥𝅰	12	1
16	G min	𝅘𝅥𝅰	16	8t	𝅘𝅥𝅰	16	33	𝅘𝅥𝅯	8	1r	𝅘𝅥𝅰	8	136r
17	A♭ maj	𝅘𝅥𝅯	4	1	𝅘𝅥𝅰	8	111	𝅘𝅥𝅯	8	6	𝅘𝅥𝅯	8	1
18	G♯ min	𝅘𝅥𝅯	6	1	𝅘𝅥𝅰	16	1	𝅘𝅥𝅯	8	1	𝅘𝅥𝅯	6	3
19	A maj	𝅘𝅥𝅯	8	2	𝅘𝅥𝅯	12	1	𝅘𝅥𝅯	6	1	𝅘𝅥𝅯	8	1
20	A min	𝅘𝅥𝅯	6	1	𝅘𝅥𝅰	16	1,21	𝅘𝅥𝅰	16	1i	𝅘𝅥𝅰	16	13r
21	B♭ maj	𝅘𝅥𝅰	16	—	𝅘𝅥𝅯	6	—	𝅘𝅥𝅯	4	s	𝅘𝅥𝅯	4	1
22	B♭ min	𝅘𝅥𝅯	8	s	𝅘𝅥𝅮	4	1	𝅘𝅥𝅮	4	14	𝅘𝅥𝅮	4	1r
23	B maj	𝅘𝅥𝅯	8	1	𝅘𝅥𝅯	8	—,1	𝅘𝅥𝅯	8	1	𝅘𝅥𝅯	8	13
24	B min	𝅘𝅥𝅯	8	55	𝅘𝅥𝅯	8	1	𝅘𝅥𝅰	8	1	𝅘𝅥𝅯	6	2

size of an artificial measure is exactly the size of a true measure. However, the purpose of having measures is to give notes measure position, in order to improve the efficiency of the search. If the size of the artificial measure is too large, some patterns will be seen as different, rather than as the same. For example, it is common to state a theme from a 4/4 piece half a true measure out of phase, such as in the D major Fugue of Book II. In the case of the C# major Prelude of Book II, there is a meter change from 4/4 to 3/8. The greatest common divisor of 16 (4/4 is same length as 16/16) and 6 (3/8 is the same length as 6/16) is 2, so the artificial measure size is 2 sixteenths.

The result column of the table is usually an integer, corresponding to the rank, by interest, at which MARPLE placed the main theme of the piece. Note that MARPLE finds other themes of lesser and sometimes greater interest, but it is more difficult to assess these systematically. There are four other symbols shown in some of the entries of the result column. An 's' means that MARPLE became computationally mired due to finding seemingly very large patterns (that human experts would discount). This is a rare event, one that we did not expect. The C major Prelude of Book I is well known, being a simple arpeggiation of a chord progression. The main theme was in some sense too prevalent, making myriad ways to build ever larger patterns. A '-' means that MARPLE did not rank the designated theme among its 200 best. A 't' indicates that MARPLE was confused by a trill, and an ‘r’ indicates that one or more rests within the theme was problematic.

There is much to be learned from these results. The rank columns show a total of 20 ‘1’s for the preludes and a total of 27 ‘1’s for the fugues. In scoring, some interpretation was involved because a pattern of many notes might differ by one or a few from that given in the book. There were five ‘s’ and ten ‘-’ for a total of 15 (of 96) pieces in which the theme was not found at all. For the remaining 34 pieces, the theme was found sometimes high in the ranked list, and sometimes far down in the list. It is tempting to repair and improve MARPLE immediately, to eliminate many of these occurrences, and these cases will be highly informative for subsequent process. However, our experiment was to assess MARPLE at the present, not to see how well we could adapt MARPLE to a given set of inputs.
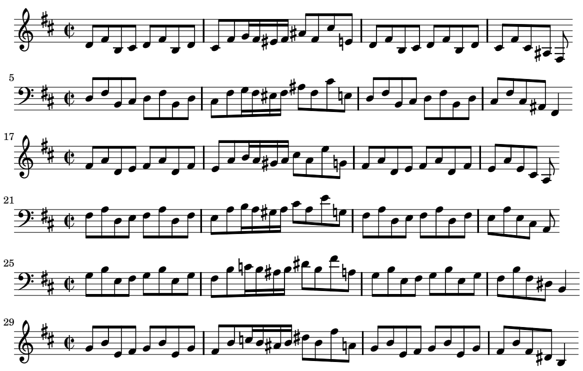
Figure 1: Prelude 24, Book II

Figure 1 shows the cluster of occurrences of the main theme found by MARPLE for the B minor Prelude from Book II. One can see by inspection of the figure that the main theme of the prelude occurs six times, at the measure numbers indicated. Notice that four of the statements are in the minor modality and that two are in the major. There is a seventh statement of the theme at measure 59, not included in the figure, but it ends with two quarter notes instead of four eighth notes like the others. MARPLE reports the group of seven themes earlier in its analysis, but as the n-grams become longer in the analysis, this seventh version becomes too different to remain in the group. It would be useful to provide a graphical depiction of the various groups (clusters) as they grow, join, and split. In any case, the present output of MARPLE shows the clusters, and renders them in music notation so that an analyst can immediately see the patterns and how to find each instance in the original score.

Two disagreements between MARPLE and the authority are noteworthy. First, in Fugue 4 of Book I, MARPLE finds a completely different prominent theme, starting at measure 36. Further analysis shows that this is a double fugue, with the second subject introduced at measure 36. The second subject is of greater interest. MARPLE also finds the first fugue subject, but it is much less interesting, being just four notes long. Second, in Fugue 15 of Book I, MARPLE identifies a shorter repetitive wedge-shaped counter subject as most interesting, and it is indeed highly prominent, also occurring more often than the subject. The main subject is identified, but it is ranked lower in the list.

Trills caused some confusion. Our initial reasoning was that trills would not score well in terms of interest, being based on a rapid alternation of just two pitches. While this is true, the number of trill patterns can become large because of the shorter trills contained within the trill. For a trill of k oscillations, there will two trills of $k-1$ oscillations, three trills of $k-2$ oscillations, and so on. Many of these will not be grouped together due to the constraint on measure position. Nevertheless, MARPLE was confused by the four occurrences of a measure long trill. This group, consisting of very long n-grams, scored very high in terms of interest. The two trills of one fewer oscillation also each occurred four times. Thus, there were a great many clusters of varying lengths that were all subtrills.

Finally, rests were problematic. In Book II, eight of the fugue subjects are punctuated by rests. One can tweak the δ parameter in the ‘consecutive’ predicate to make pairs of notes that may seem distant (in time) satisfy the test for being consecutive. When this is done, many more note pairs are admitted to the initial set of 2-grams. This causes the algorithm to run much more slowly, sometimes too slowly. A better design is needed for this thorny problem.

One can of course debate some of the musical calls by the authority. Two are worth mentioning. In the F# major Prelude of Book II, MARPLE finds the theme that starts at measure 4 to be prevalent and of high interest. The authority seems just to show the opening few measures in every case. A listener would likely find the later theme more noticeable, but this is simply our opinion.

Similarly, the A minor Prelude of Book II is a duet, with two highly distinctive themes. MARPLE finds both themes, ranking one at #1 and the other at #21. The authority lists just one of the themes.

7 Conclusions

We addressed the problem of how to construct a practical algorithm for identifying salient note patterns in polyphonic music represented as a MIDI file. To do this, we identified the computational issues, and showed how knowledge of the domain could be employed to render the search tractable. The result is the MARPLE algorithm, implemented as a computer program. Sufficient detail has been provided for it to be reproduced. We proceeded to evaluate MARPLE on a set of 96 specific pieces that were not used at any time for MARPLE's development. To our knowledge, this is the first systematic analysis of this kind of pattern finder. It is important for scientific discourse and progress to make such assessments that report specific strengths and weaknesses. Our findings are generally that MARPLE often succeeds at its task. It runs quickly, measured in seconds and sometimes minutes. For some pieces, in which very large patterns could be construed, MARPLE can become mired. MARPLE can be confused by rests that occur within a theme, and can become distracted by trills. Both the strengths and weaknesses of the present MARPLE shed light on the issues and how to direct subsequent inquiry.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant Number 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. We thank Stephen Murtagh and Dennis Gove for comments on a draft. The comments of the reviewers were most helpful. The MIDI files were made available through the Classical Music Archive.

References

Barlow, H., & Morgenstern, S. (1948). *A dictionary of musical themes*. New York: Crown Publishers, Inc..

Cambouropoulos, E., Crochemore, M., Iliopoulos, C.S., Mouchard, L., & Pinzon, Y.J. (2002). Algorithms for computing approximate repetitions in musical sequences. *International Journal of Computational Mathematics, 79*, 1135-1148.

Conklin, D. (2002). Representation and discovery of vertical patterns in music (pp. 32-42). In Anagnostopoulou, Ferrand & Smaill (Eds.), *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Jan, S. (2004). Meme hunting with the Humdrum Toolkit: Principles, problems, and prospects. *Computer Music Journal, 28*, 68-84.

Kirlin, P. B., & Utgoff, P. E. (2005). Learning to segregate voices in explicit and implicit polyphony. *Proceedings of the Sixth International Conference on Music Information Retreival* (pp. 552-557).

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.

Lartillot, O. (2004). A musical pattern discovery system founded on a modeling of listening strategies. *Computer Music Journal, 28*, 53-67.

Lartillot, O. (2005). Efficient extraction of closed motivic patterns in multi-dimensional symbolic representations of music. *Proceedings of the International Conference on Music Information Retrieval* (pp. 191-198).

Lerdahl, F., & Jackendoff, R. (1983). *A generative theory of tonal music*. Cambridge, MA: MIT Press.

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

Meredith, D., Lemström, K., & Wiggins, G.A. (2002). Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research, 31*, 321-345.

Patel, N., & Mundur, P. (2005). An n-gram based approach to finding the repeating patterns in musical data. *Proceedings of the European IMSA*. Grindelwald, Switzerland.

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

Rolland, P.-Y. (2001). FlExPat: Flexible extraction of sequential patterns. *Proceedings of the International Conference on Data Mining* (pp. 481-488).

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.