
Journal of Intelligent Information Systems, 21:1, 9–33, 2003

Automatic Thematic Extractor

COLIN MEEK* meek@umich.edu
WILLIAM P. BIRMINGHAM
University of Michigan, Electrical Engineering and Computer Science Dept., 1101 Beal Ave., Ann Arbor, MI 48109, USA

Received June 15, 2002; Revised August 5, 2002; Accepted August 15, 2002

Abstract. We have created a system that identifies musical "keywords" or themes. The system searches for all patterns composed of melodic (intervallic for our purposes) repetition in a piece. This process generally uncovers a large number of patterns, many of which are either uninteresting or only superficially important. Filters reduce the number or prevalence, or both, of such patterns. Patterns are then rated according to perceptually significant characteristics. The top-ranked patterns correspond to important thematic or motivic musical content, as has been verified by comparisons with published musical thematic catalogs. The system operates robustly across a broad range of styles, and relies on no meta-data on its input, allowing it to independently and efficiently catalog multimedia data.

Keywords: music information retrieval, cataloging, metadata creation, music

1. Introduction

We are interested in extracting the major themes from a musical piece: recognizing patterns and motives in the music that a human listener would most likely retain. "Thematic extraction," as we term it, has interested musician and AI researchers for years. Music librarians and music theorists create thematic indices (e.g., Köchel catalog (Köchel, 1978)) to catalog the works of a composer or performer. Moreover, musicians often use thematic indices (e.g., Barlow's A Dictionary of Musical Themes (Barlow, 1975)) when searching for pieces (e.g., a musician may remember the major theme, and then use the index to find the name or composer of that work). These indices are constructed from themes that are manually extracted by trained music theorists. Construction of these indices is time consuming and requires specialized expertise. Figure 1 shows a simple example.

The best known methods for automated thematic extraction require some "hand tweaking" (Cope, 1996) to at least provide clues about what a theme may be, or generate thematic listings based solely on repetition and string length (Alexandra, 1998; Tseng, 1999). Two music information retrieval systems (Chai, 2001; Lu, 2001) identify important tracks (or "voices") in a MIDI file based on the track label. This has proven effective for pop music, where tracks containing melodies are frequently indicated "vocal" for instance. Yet, automatically extracting major themes is an extremely important problem to solve. In addition to aiding music librarians and archivists, exploiting musical themes is key to developing

*Author to whom all correspondence should be addressed.

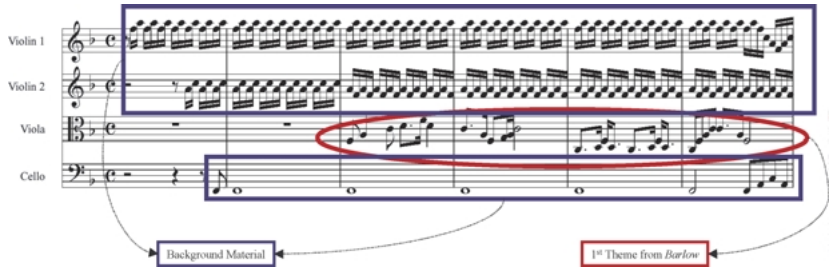

Figure 1. Sample thematic extraction from opening of Dvorak's *American Quartet*.

efficient music-retrieval systems. The reasons for this are twofold. First, it appears that themes are a highly attractive way to query a music-retrieval system. Second, because themes are much smaller and less redundant than full pieces, by searching a database of themes, we simultaneously get faster retrieval (by searching a smaller space) and get increased relevancy. Relevancy is increased as only crucial elements, variously named "motives," "themes," "melodies," or "hooks," are searched, thus reducing the chance that less important, but commonly occurring, elements will fool the system.

There are many aspects to music, such as melody, structure and harmony, each of which may affect what we perceive as major thematic material. Extracting themes is a difficult problem for many reasons. Among these are the following:

– The major themes may occur anywhere in a piece. Thus, one cannot simply scan a specific section of piece (e.g., the beginning).
– The major themes may be carried by any voice. For example, in figure 2, the viola, the third lowest voice, carries the principal theme. Thus, one cannot simply "listen" to the upper voices.

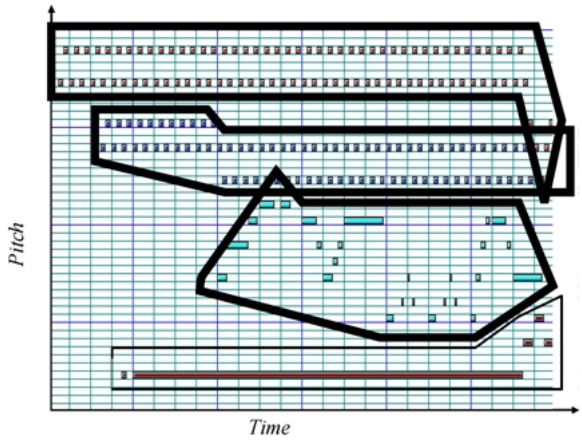

Figure 2. Opening phrase of Dvorak's *American Quartet*.

– There are highly redundant elements that may appear as themes, but should be filtered out. For example, scales are ubiquitous, but rarely constitute a theme. Thus, the relative frequency of a series of notes is not sufficient to make it a theme.

In this paper, we introduce an algorithm, Melodic Motive Extractor (MME), that automatically extracts themes from a piece of music, where music is in a "note" representation. Pitch and duration information are given; metrical and key information is not required.

MME exploits redundancy that is found in music: composers will repeat important thematic material. Thus, by breaking a piece up into note sequences and seeing how often sequences repeat, we identify the themes. Breaking up involves examining all note sequences of length two to some constant. Moreover, because of the problems listed earlier, we must examine the entire piece and all voices. This leads to very large numbers of sequences (roughly 7000 sequences on average, after filtering), thus we must use a very efficient algorithm to compare these sequences.

Once repeating sequences have been identified, we must further characterize them with respect to various perceptually important features in order to evaluate if the sequence is a theme. Learning how best to weight these features for the thematic value function is an important part of our work. For example, we have found that the frequency of a pattern is a stronger indication of thematic importance than is the register in which the pattern occurs (a counterintuitive finding). We implement hill-climbing techniques to learn weights across features. The resulting evaluation function then rates the sequences.

Across a corpus of 60 works, drawn from the Baroque, classical, romantic and contemporary periods, MME extracts sections identified by Barlow as "1st themes" over 98% of the time.

2. Problem formulation

Input to MME is a set of note events making up a musical composition $N = \{n_1, n_2, \ldots, n_3\}$. A note event is a triple consisting of an onset time, an offset time and a pitch (in MIDI note numbers, where 60 = "Middle C" and the resolution is the semi-tone): $n_i = \langle onset_i, offset_i, pitch_i \rangle$. We note that several other valid representations of a musical composition exists, taking into account amplitude, timbre, meter and expression markings among others (Simoni, 2000). We limit the domain because pitch is reliably and consistently stored in MIDI files—the most easily accessible electronic representation for music—and because we are interested primarily in voice contour as a measure of redundancy.

The goal of MME is to identify patterns and rank them according to their perceptual importance as a theme. We readily acknowledge that there may, in some cases, be disagreement among listener about what constitutes a theme in a piece of music; however, we note that the published thematic catalogs represent common convention. These catalogs thereby provide a concrete measure by which the system can be evaluated.

3. Algorithm

In this section, we describe the operation of MME. This includes identifying patterns and computing pattern characteristics, such that "interesting" patterns can be identified.

MME's main processing steps are the following:

– Input
– Register
– Stream segregation
– Filter top voice
– Calculate event transitions
– Generate event keys
– Identify and filter patterns
– Frequency
– Compute other patter features
– Rate patterns
– Return results

3.1. Input

MME generally takes as input MIDI files, which are translated into lists of note events in the described format. Information is also maintained about the channel and track of each event, which is used to separate events into "streams."

3.2. Register

Register is an important indicator of perceptual prominence (Bregman, 1990): we listen for higher pitched material. For the purposes of MME, we define register in terms of the "voicing", so that for a set of n concurrent note events, the event with the highest pitch is assigned a register of 1, and the event with the lowest pitch is assigned a register value of n. For consistency across a piece, we map register values to the range [0, 1] for any set of concurrent events, such that 0 indicates the highest pitch, 1 the lowest.

We need to define the notion of concurrency more precisely. Two events with time intervals $I_1 = [s_1, e_1]$ and $I_2 = [s_2, e_2]$ are considered concurrent if there exists a common interval $I_c = [sc, ec]$ such that $sc < ec$ and $I_c \subseteq I1 \wedge Ic \subseteq I_2$. The simplest way of computing these values is to walk through the event set ordered by onset time, maintaining a list of active events (notes that are on or sounding), or events sharing a common interval. We assign to notes sounding alone a value of 0, since although such notes are both the highest *and* lowest, they are (trivially) the most significant occurrences at a particular time. Since the register value can change over the course of a note, the register value for a note is arbitrarily set to the maximum of all values (see figure 3).

3.3. Stream segregation and filtering top voice

Generally, the individual channels of a MIDI file correspond to the different instruments or voices of a piece. Figure 1 shows a relatively straightforward example of segmentation, from the opening of Dvorak's "American" Quartet, where four voices are present. In cases where several concurrent voices are present in one instrument, for example in piano music, we deal

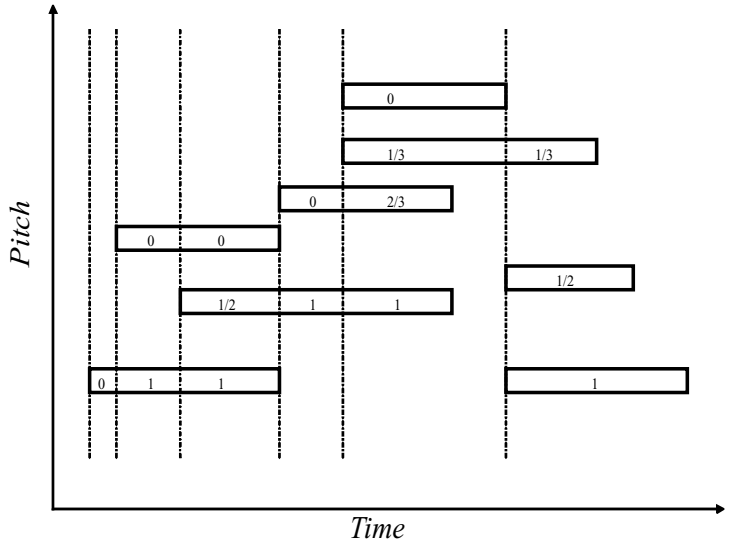

Figure 3. Register, Example Piece.

with only the top sounding voice. This is clearly a restriction, albeit a reasonable one, as certain events are disregarded. This restriction is necessary. Although existing analysis tools, such as MELISMA (Temperley, 1999), perform stream segregation on abstracted music, i.e., note-event representation, they have trouble with overlapping voices (Temperley, 2000), as seen between the middle voices in figure 1.

Identifying the top sounding voice is not as straightforward as it may appear. Some MIDI scores contain overlapping consecutive events within a single voice. To avoid filtering out such notes, we employ an algorithm similar to the register algorithm, wherein events are removed from the active list for their particular channel after some ratio (k) of their duration from their onset, and as such avoid being falsely labeled as "lower-sounding" notes. We have found that $k = 0.5$ provides solid performance. For instance, an event in the time interval [30, 50] will be removed from the active list when the sweep reaches time $40 = 30 + (50 - 30)\,k$.

Additionally, when long pauses (greater than some time constant) are found in a "stream," the stream is broken at that point, and a new stream is created. In this manner, we exclude sequences enclosing large stretches of silence from gaining arbitrary advantage from the "duration" feature.

For the purposes of this paper, we will henceforth indicate events using the notation $e_{stream,index}$, such that $e_{1,2}$ indicates the second note of the first stream. Similarly, we will use the notation $pitch_{stream,index}$ to refer to the pitch of the event at the given position.

3.4. Calculating transitions

We are primarily concerned with melodic contour as an indicator of redundancy. For our purposes, contour is defined as the sequence of pitch intervals across a sequence of note

events in a stream. For instance, the stream consisting of the following event sequence: $\{\langle 0, 1, 60\rangle, \langle 1, 2, 62\rangle, \langle 2, 3, 64\rangle, \langle 3, 4, 62\rangle, \langle 4, 5, 60\rangle\}$ has contour $\{+2, +2, -2, -2\}$.

MME considers contour in terms of "simple interval," which means that although the sign of an interval $(+/-)$ is considered, octave is not. As such, an interval of $+2$ is equivalent to an interval of $+14 = (+2 + \text{octave} = +2 + 12)$. We normalize each interval corresponding to an event, i.e., the interval between that event and its successor, to the range $[-12, 12]$:

$$\begin{aligned} interval_{s,i} &= Pitch[e_{s,i+1}] - Pitch[e_{s,i}] \\ \text{Simple interval} = c_{s,i} &= \begin{cases} interval_{s,i} & \text{if } -12 \le interval_{s,i} \le 12 \\ -mod_{12}(-interval_{s,i}) & \text{if } interval_{s,i} < -12 \\ mod_{12}(interval_{s,i}) & \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

Another transition measure we employ is known as the Inter-Onset Interval (IOI), used to describe the rhythmic content of a sequence, and the rhythmic consistency of a pattern. This measure ignores the rhythmic articulation of events, but maintains the basic rhythmic information. In the stream $\{\langle 0, 1, 60\rangle, \langle 1, 2, 62\rangle, \langle 2, 3, 64\rangle, \langle 3, 4, 62\rangle, \langle 4, 5, 60\rangle\}$, the IOI values are $\{1, 1, 1, 1\}$, where:

$$IOI_{s,i} = onset_{s,i+1} - onset_{s,i} \tag{2}$$

3.5. Calculating keys

To efficiently uncover patterns, or repeating sequences, we assign an integer key $k_{s,i}$ to each event in the piece that uniquely identifies a sequence of M intervals, where M is the maximum number of intervals we consider in a pattern. The notation $k_{s,i}(m)$ represents an m-length key assigned to stream s at index i. The keys must exhibit the following property:

$$k_{s1,i1} = k_{s2,i2} \leftrightarrow \{c_{s1,i1}, c_{s1,i1+1}, \ldots, c_{s1,i1+m-1}\} = \{c_{s2,i2}, c_{s2,i2+1}, \ldots, c_{s2,i2+m-1}\} \tag{3}$$

Simply put, the key must completely and uniquely represent the underlying intervallic sequence up to the indicated length. Since only 25 distinct simple intervals exist, we can refer to sequences of intervals in radix-26 notation, reserving a digit (0) for the ends of streams. An m-digit radix-26 number, where each digit corresponds to an interval in sequence, thus uniquely identifies that sequence of intervals, and our key values can then be calculated as follows, re-mapping intervals to the range [1, 25]:

$$k_{s,i}(m) = \sum_{j=0}^{m-1} (c_{s,i+j} + 13) \cdot 26^{m-j-1} \tag{4}$$

The following derivations allow us to more efficiently calculate the value of $k_{s,i}$:

$$k_{s,i}(1) = c_{s,i} + 13 \tag{5}$$

$$k_{s,i}(m) = \begin{cases} 26 \cdot k_{s,i}(n-1) + k_{s,i+n-1}(1) & \text{if } i+n < |c_s| \\ k_{s,i}(|c_s| - i)) \cdot 26^{n-|c_s|+i-1} & \text{otherwise} \end{cases}, \tag{6}$$

where $|c_s|$ is the length of the stream.

The second case of this last equation deals with the situation where no additional information is gained by increasing n, since there are no additional intervals to consider beyond the end of the stream. It is derived from the observation that when $i > |c_s|$, $k_{s,i}(1) = 0$, the end of stream zero padding.

By removing the most significant digit of a key $k_{s,i}(n)$, we get the key for the subsequent event $k_{s,i+1}(n-1)$:

$$k_{s,i+1}(n-1) = k_{s,i}(n) - k_{s,i}(1) \cdot 26^{n-1} \tag{7}$$

We can therefore calculate the subsequent key value in constant time, using Eqs. (6) and (7).

Using Eqs. (5) and (6), we can calculate the value of $k_{s,1}$ in linear time with respect to the maximum pattern length, or the stream length, whichever is smaller (this is essentially an application of Horner's Rule (Rivest, 1999)). Equation (3) allows us to calculate the key of each subsequent event in stream s in constant time (as with the Rabin-Karp algorithm (Rivest, 1999)). As such, the overall complexity for calculating keys is $\Theta(n)$ with respect to the number of events.

Consider the following simple example for $M = 4$, a single phrase from Mozart's *Symphony no. 40*: $c_1 = \{-1, 0, +1, -1, 0, +1, -1, 0, +8\}$.

First we calculate the key value for the first event ($k_{1,1}(4)$), using Eqs. (5) and (6) recursively:

$$\begin{aligned} k_{1,1}(4) &= 26 \cdot k_{1,1}(3) + k_{1,4}(1) \quad \text{(Eq. (6))} \\ &= 26 \cdot (26 \cdot k_{1,1}(2) + k_{1,3}(1)) + 12 \quad \text{(Eqs. (5) and (6))} \\ &= 26 \cdot (26 \cdot (26 \cdot k_{1,1}(1) + k_{1,2}(1)) + 14) + 12 \\ &= 26 \cdot (26 \cdot (26 \cdot 12 + 13) + 14) + 12 \\ &= 220076 \end{aligned}$$

Then we calculate the remaining key values:

$$\begin{aligned} k_{1,2}(3) &= k_{1,1}(4) - k_{0,0}(1) \cdot 26^3 = 9164 \quad \text{(Eq. (7))} \\ k_{1,2}(4) &= 26 \cdot k_{1,2}(3) + k_{0,4}(1) = 238277 \quad \text{(Eq. (6))} \end{aligned}$$

Using the same procedure, we generate the remaining key values:

$$\begin{aligned} k_{1,3}(4) = 254528,\ k_{1,4}(4) = 220076,\ k_{1,5} = 238277,\ k_{1,6}(4) = 254535, \\ k_{1,7}(4) = 220246,\ k_{1,8}(4) = 242684,\ k_{1,9} = 369096,\ k_{1,10}(4) = 0 \end{aligned}$$

3.6. *Identifying and filtering patterns*

We employ one final derivation on k for the pattern identification:

$$\forall m, 1 \leq m \leq M : \left\lfloor \frac{k_{s,i}(M)}{26^{M-m}} \right\rfloor \tag{8}$$

This implies we can "recover" shorter m-length keys from an M-length key. Events are then sorted on key so that occurrences of a particular pattern are adjacent in the ordering. We make a pass through the list for pattern lengths from $m = \{M, M-1, \ldots, 2\}$, resulting in a set of patterns, ordered from longest to shortest. This procedure is straightforward: during each pass through the list, we group together keys for which the value of $k(m)$—calculated using Eq. (8)—is the same. Such groups are consecutive in the sorted list. Occurrences of a given pattern are then ordered according to their onset time, a property necessary for later operations.

Returning to our Mozart example, sorting the keys we get: $\{k_{1,10}, k_{1,1}, k_{1,4}, k_{1,7}, k_{1,2}, k_{1,5}, k_{1,8}, k_{1,3}, k_{1,6}, k_{1,9}\}$. On our first pass through the list, for $m = 4$, we identify patterns $\{k_{1,1}, k_{1,4}\}$ and $\{k_{1,2}, k_{1,5}\}$, since their keys are identical. During the second pass, for $m = 3$, we identify patterns $\{k_{1,1}, k_{1,4}\}$, $\{k_{1,2}, k_{1,5}\}$ and $\{k_{1,3}, k_{1,6}\}$, noting that $\lfloor \frac{k_{1,3}}{26^{4-3}} \rfloor = \lfloor \frac{k_{1,6}}{26^{4-3}} \rfloor$ (which by Eq. (8) indicates that a pattern of length three exists.) Similarly, we identify the following patterns for $m = 2$: $\{k_{1,1}, k_{1,4}, k_{1,7}\}$, $\{k_{1,2}, k_{1,5}\}$ and $\{k_{1,3}, k_{1,6}\}$. The patterns are shown in Table 1. We associate a vector of parameter values (notated $C_i = \langle v_1, v_2, \ldots, v_n \rangle$) and a set of occurrences to each pattern: each pattern is quantified according to various features which will be described later, and we maintain information about where instances of the pattern occur. Length, v_{length}, is one such parameter. The assumption was made that longer patterns are more significant, simply because they are less likely to occur by chance.

As patterns are identified, they are filtered according to several criteria. Since zero padding is used at the ends of streams, it must be verified that a sequence does not overrun the end of a stream, which frequently happens since all streams end with the same

Table 1. Patterns in opening phrase of Mozart's *Symphony no. 40*.

Pattern	Occurrences at	Characteristic interval sequence
P_1	$e_{1,1}, e_{1,4}$	$\{-1, 0, +1, -1\}$
P_2	$e_{1,2}, e_{1,5}$	$\{0, +1, -1, 0\}$
P_3	$e_{1,1}, e_{1,4}$	$\{-1, 0, +1\}$
P_4	$e_{1,2}, e_{1,5}$	$\{0, +1, -1\}$
P_5	$e_{1,3}, e_{1,6}$	$\{+1, -1, 0\}$
P_6	$e_{1,1}, e_{1,4}, e_{1,7}$	$\{-1, 0\}$
P_7	$e_{1,2}, e_{1,5}$	$\{0, +1\}$
P_8	$e_{1,3}, e_{1,6}$	$\{+1, -1\}$

zero-padding. Two other filtering criteria are considered as well: intervallic variety, and doublings.

Calculating keys takes $\Theta(n)$ time with respect to the number of note events in the piece. Identifying patterns is dominated by the sorting of events based on key, but the maximum pattern length under consideration (M) can become a significant variable, so the identification phase runs in $\Theta(n \log n + Mn)$ time. For the purposes of the complexity analyses, M and n will refer to maximum pattern length and the number of notes in the input piece respectively throughout this paper.

3.7. Intervallic variety

Early experiments with this system indicated that sequences of repetitive, simple pitch-interval patterns dominate given the parameters outlined thus far. For instance, in the Dvorak example (see figure 2) the melody is contained in the second voice from the bottom, but highly consistent, redundant figurations exist in the upper two voices. Intervallic variety provides a means of distinguishing these two types of line, and tends to favor important thematic material since that material is often more varied in terms of contour.

Given that intervallic variety is a useful indicator of how interesting a particular passage appears, we count the number of distinct intervals observed within a pattern, not including 0. We calculate two interval counts: one in which intervals of $+x$ or $-x$ are considered equivalent, the other taking into account interval direction. Considering the entire Mozart example, which is indeed a pattern within the context of the whole piece, there are three distinct directed intervals, -1, $+1$ and 8, and two distinct undirected intervals, 1 and 8.

At this stage, we filter out all patterns whose characteristic interval sequence has below certain minimum values for these interval counts. In addition, interval counts are maintained for each pattern.

The input piece contains, worst-case, $\Theta(Mn)$ patterns, and for each of these patterns, it takes $\Theta(M)$ time to compute the intervallic variety value, leading to an overall complexity of $\Theta(M^2n)$. Note, however, that by maintaining detailed interval counts for given pattern lengths beginning at successive events, we can achieve $\Theta(Mn)$ performance overall. This is based on the observation that given the specific interval counts for the interval sequence $\{c_i, c_{i+1}, \ldots, c_{i+m-1}\}$ we can straightforwardly calculate the intervals count for the interval sequence $\{c_{i+1}, c_{i+2}, \ldots, c_{i+m}\}$ in constant time, with reference to c_i and c_{i+m} only.

3.8. Doublings

Doublings are a special case in MME. A "doubled" passage occurs where two or more voices simultaneously play the same line. In such instances, only one of the simultaneous occurrences is retained for a particular pattern, the highest sounding to maintain the accuracy of the register measure.

We must provide a definition of simultaneity to clearly describe this parameter. To provide for inexact performance, we allow for a looser definition: two occurrences of a pattern at $e_{s1,i1}$ and $e_{s2,i2}$ with length m, are considered simultaneous if and only if $\forall j, 0 \leq j \leq m : e_{s1,i1+j}$ overlaps $e_{s2,i2+j}$. Two events are in turn considered overlapping if they strictly intersect. It is easier to check for the non-intersecting relations—using the conventions and notations of Beek (Beek, 1996)—$e_{s1,i1}$ before (b) $e_{s2,i2}$ or the inverse (bi) (see Algorithm 1). This doubling filtering occurs before other computations, and thus influences frequency. We do, however, retain the doubling information, as it is a musical emphasis technique. If after filtering doublings less than two occurrences remain, the pattern is no longer considered a pattern, and is removed from consideration. Doublings serve to reinforce a voice, and as such do not constitute repetition.

This process runs in worst-case $O(k^2M)$ time per pattern, where k is the number of occurrences in the pattern. There can be no more than M occurrences of patterns beginning

Algorithm 1 Filter doublings

Given an m-length pattern P with n occurrences at $e_{s_1,i_1}, e_{s_2,i_2}, \ldots, e_{s_n,i_n}$ and with reference to two n element boolean arrays *Remove* and *Doubled* initialized to **false**:

```
1: for j ← 1 to n − 1 do
2:   for k ← j + 1 to n do
3:     if ¬Remove[j] and ¬Remove[k] then
4:       Simul ← true
5:       for l ← 0 to m do
6:         if ¬Intersects(e_{s_j,i_j+l}, e_{s_k,i_k+l}) then
7:           Simul ← false
8:           l ← m + 1
9:         end if
10:        if Simul then
11:          if pitch_{s_j,i_j+l} > pitch_{s_k,i_k+l} then
12:            Remove[k] ← true
13:            Doubled[j] ← true
14:          else
15:            Remove[j] ← true
16:            Doubled[k] ← true
17:          end if
18:        else
19:          k ← n + 1
20:        end if
21:      end for
22:    end if
23:  end for
24: end for
25: remove occurrences where Remove is true
```

at each event (one occurrence of each length), so given l patterns with occurrence counts $k_1, k_2, \ldots, k_l$, we have the following restriction: $\sum_{i=1}^{l} k_i = \mathrm{O}(Mn)$. So the worst case maximizes $\sum_{i=1}^{l} {k_i}^2 M$, which is the case where a single pattern exists for each distinct length, with a complexity of $\mathrm{O}(n^2 M)$. Note that experimentally MME exhibits sub-linear complexity (see figure 11), taking advantage of anecdotally observed trends in input data. Note also that when there are no doublings as is most often the case, the doubling verification runs in $\Theta(k)$ time for each pattern.

3.9. Frequency

Frequency of occurrence is one of the principal parameters considered by MME in establishing pattern importance. All other things being equal, higher occurrence frequency is considered an indicator of higher importance. Our definition of frequency is complicated by the inclusion of partial pattern occurrences. For a particular pattern, characterized by the interval sequence $\{c_1, c_2, \ldots, c_{v_{length}}\}$, the frequency of occurrences is defined as follows:

$$\left[\sum_{m=v_{length}}^{2} \sum_{j=1}^{v_{length}-m+1} \left(\begin{array}{c} \text{non-redundant occurrences of} \\ \{c_j, c_{j+1}, \ldots, c_{j+m-1}\} \end{array} \right) \cdot m \right] \Big/ v_{length} \tag{9}$$

An occurrence is considered non-redundant if it has not already been counted, or partially counted (i.e., it contains part of another sub-sequence that is longer or precedes it.) Consider the piece consisting of the following interval sequence, in the stream e_1: $c_1 = \{-2, +2, -2, +2, -5, +5, -2, +2, -2, +2, -5, +5, -2, +2, -2, +2\}$, and the pattern $\{-2, +2, -2, +2, -5\}$. Clearly, there are two complete occurrences at $e_{1,1}$ and $e_{1,7}$, but also a partial occurrence of length 4 at $e_{1,13}$. The frequency is 2.8 for this pattern. In practice, we avoid double-counting pattern occurrences by tagging the underlying events with unique pattern identifiers (see Algorithm 3).

To efficiently calculate frequency, we first construct a set of pattern occurrence lattices, on the following binary occurrence relation (denoted $\prec$): Given occurrences o_a and o_b characterized by event sequences E_a and E_b, $o_a \prec o_b$ if and only if $E_a \subset E_b$. In other words, each occurrence in the lattice covers all subsequences.

As such, in establishing frequency, we need consider only those patterns covered by occurrences of that pattern in the lattices. Two properties of our data facilitate this construction:

1. The pattern identification procedure adds patterns in reverse order of pattern length.
2. For any pattern occurrence of length $m > 2$, there are at most two occurrences of length $m - 1$, one sharing the same initial event, one sharing the same final event. If one of these two child occurrences does not exist, it is due to the filtering described above. Because of the nature of the filtering, no patterns of length less than $m - 1$ will be covered by the occurrence in these instances, so we need only generate links to

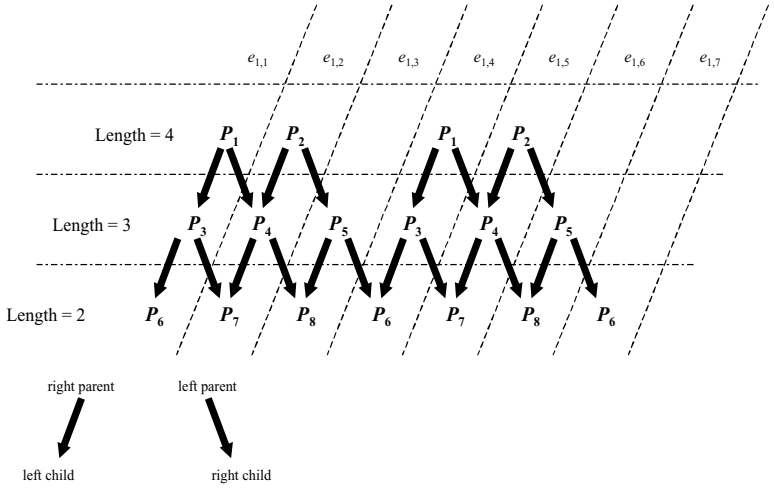

Figure 4. Lattice for the first phrase of Mozart's *Symphony no. 40*.

occurrences of length $m - 1$ in the lattices. The branching factor is thus limited to two.

The lattice is described as follows: given a node representing an occurrence (o) of a pattern with length m, the left child is an occurrence of length $m - 1$ beginning at the same event. The right child is an occurrence of length $m - 1$ beginning at the following event. The left parent is an occurrence of length $m + 1$ beginning at the previous event, and the right parent is an occurrence of length $m + 1$ beginning at the same event.

Consider the patterns in the Mozart excerpt (see Table 1): P_1's first occurrence, with length 4 and at $e1, 1$ directly covers two other occurrences of length 3: P_3's first occurrence at $e1, 1$ (left child) and P_4's first occurrence at $e_{1,2}$ (right child). The full lattice is shown in figure 4, where each occurrence in the lattice is labeled with its respective pattern.

Lattices are constructed from the top down, since patterns are added in reverse order of length. We maintain an array of pointers to occurrences, with an entry for each event in the piece, so that as occurrences are added, lattice links can be built in constant time (see Algorithm 2).

Consider the patterns identified in the short Mozart example (Table 1), from which we build the lattice in figure 4. In this example, patterns are added in order of index: $P_1, P_2, P_3, \ldots$ When the first occurrence of pattern P_5 is inserted, o_{left} = the first occurrence of P_4, and o_{right} = null. Since P_4 has the same length as P_5, we check the right parent of o_{right}, and update the link between those occurrences of P_2 and P_5. Other links are updated in a more straightforward manner. From the lattice, we easily identify non-redundant partial occurrences of patterns. For each pattern, we perform a breadth-first traversal from

its occurrences in the lattice, marking patterns and events as they are counted so that none are included twice. Simultaneously, the number of doubled occurrences is counted. In this manner, we calculate the value of the $v_{doublings}$ and $v_{frequency}$ features for each pattern (see Algorithm 3).

Algorithm 2 Lattice Construction

An occurrence o_x has fields *LeftChild*, *RightChild*, *Length*, *Pattern*, *ID*, *Stream*, *Index*, and *LeftParent*. Fields are denoted $o_x.\ Field$ in the pseudo-code. Given a sequence of patterns $P_1, P_2, \ldots, P_n$ in descending order of pattern length, and an array of pointers to occurrences ($ptr_{stream,index}$) associated with piece events initialized to **null**:

```
 1: for i ← 1 to n do
 2:    {o_1, o_2, ..., o_k} ← all occurrences of P_i
 3:    for j ← 1 to k do
 4:       {get the stream and index where occurrence begins}
 5:       stream ← o_j.stream
 6:       index ← o_j.index
 7:       {occurrence pointed to by the first event of o_j}
 8:       o_right ← ptr_stream,index
 9:       {occurrence pointed to by the event preceding o_j}
10:       if index = 1 then
11:          o_left ← null
12:       else
13:          o_left ← ptr_stream,index−1
14:       end if
15:       {we consider three cases for the value of o_left}
16:       if o_left =null then
17:          {we learn nothing about the lattice}
18:       else if o_left.length > o_j.length then
19:          o_left.RightChild ← o_j
20:       else
21:          o_left.RightParent.RightChild ← o_j
22:       end if
23:       {we consider two cases for the value of o_right}
24:       if o_right = null then
25:          {we learn nothing about the lattice}
26:       else
27:          o_j.RightParent ← o_right
28:          o_right.LeftChild ← o_j
29:          ptr_stream,index ← o_j
30:       end if
31:    end for
32: end for
```

Algorithm 3 Calculating Frequency

Patterns and events have a field *ID* which allows them to be tagged if they have been counted towards the frequency of a certain pattern. The properties of a pattern are denoted p_{v_i}. See Algorithm 2 for a list of the fields associated with occurrences. Given a pattern P, and a pattern queue Q:

1: $id \leftarrow$ unique identifier for pattern P
2: $P.ID \leftarrow id$
3: **enqueue**(Q, P)
4: **while** $\neg empty(Q)$ **do**
5: **dequeue**(Q, p) {get current pattern p}
6: $o_{current} \leftarrow$ the first occurrence of pattern p
7: $o_{left} \leftarrow o_{current}.LeftChild$
8: $o_{right} \leftarrow o_{current}.RightChild$
9: **if** $o_{left} \neq$ **null and** $o_{left}.Pattern.ID \neq id$ **then**
10: $o_{left}.Pattern.ID \leftarrow id$
11: **enqueue**(Q, $o_{left}.Pattern$)
12: **end if**
13: **if** $o_{right} \neq$ **null and** $o_{right}.Pattern.ID \neq id$ **then**
14: $o_{right}.Pattern.ID \leftarrow id$
15: **enqueue**(Q, $o_{right}.Pattern$)
16: **end if**
17: {count non-redundant occurrences of p}
18: **for** $i \leftarrow 1$ **to** the number of occurrences **do**
19: $o_{current} \leftarrow i^{\text{th}}$ occurrence of p
20: **if** no events in $o_{current}$ have $e_{stream,index}.ID = id$ **then**
21: for all events in $o_{current}$, set $e_{stream,index}.ID \leftarrow id$
22: $p_{v_{frequency}} \leftarrow p_{v_{frequency}} + o_{current}.Length$
23: **if** $o_{current}$ is doubled **then**
24: $p_{v_{doublings}} \leftarrow p_{v_{doublings}} + o_{current}.Length$
25: **end if**
26: **end if**
27: **end for**
28: **end while**

Take for instance pattern P_3 in the Mozart example. By breadth-first traversal, starting from either occurrence of P_3, the following elements are added to Q: P_3, P_6 and P_7. First, we add the two occurrences of P_3, tagging events $e_{1,1,\ldots,6}$, and setting $v_{frequency} \leftarrow 6$. The first two occurrences of P_6 contain tagged events, so we reject them, but the third occurrence at $e_{1,7}$ is un-tagged, so we tag events $e_{1,7}$ through $e_{1,9}$ and set $v_{frequency} \leftarrow 6 + 2$. All occurrences of P_7 are now tagged, so the total frequency of P_3 is equal to $\frac{8}{3}$.

This stage has a worst-case time complexity of $\Theta(M^3n^2)$, since each pattern (of which there are in the worst case $\Theta(Mn)$) covers $\Theta(M^2)$ patterns in the lattice, each of which consists of at most n occurrences, which must be checked. Similarly, a maximum of n

events can then be tagged for each pattern. We emphasize that this analysis reflects the pathological case.

3.10. Other pattern features

Several pattern features have been described thus far: $v_{interval_{count}}$, $v_{absolute_{interval_{count}}}$, v_{length}, $v_{frequency}$ and $v_{doublings}$. In addition, we consider pattern duration ($v_{duration}$), rhythmic consistency (v_{rhythm}), position in the piece ($v_{position}$), and register (calculated from event register, $v_{register}$).

3.10.1. Duration. The duration parameter is an indicator of the temporal interval over which occurrences of a pattern exist. For a given occurrence o, with initial event $e_{s1,i1}$ and final event $e_{s2,i2}$, the duration $Dur(o) = offset_{s2,i2} - onset_{s1,i1}$. For a pattern P, with occurrences $o_1, o_2, \ldots, o_k$, the distance parameter is the average duration of all occurrences:

$$v_{duration} = \left[\sum_{i=1}^{k} Dur(o_i)\right] \Big/ k \tag{10}$$

3.10.2. Rhythmic consistency. We calculate the rhythmic distance between a pair of occurrences as the angle difference between the vectors built from the IOI values of each occurrence. We represent the rhythm of an occurrence (o) as a vector comprised of its IOI values $\bar{V}(o)$. The rhythmic distance between a pair of occurrences o_a and o_b is then the angle distance between the vectors $\bar{V}(o_a)$ and $\bar{V}(o_b)$:

$$Dist(o_a, o_b) = \cos^{-1}\left(\frac{\overbrace{\bar{V}(o_a) \cdot \bar{V}(o_b)}^{\text{vector dot product}}}{\underbrace{\left\|\bar{V}(o_a)\right\| \left\|\bar{V}(o_a)\right\|}_{\text{product of vector magnitudes}}}\right) \tag{11}$$

A 3-dimensional example of the rhythmic distance calculation between two occurrences is shown in figure 5. We take the average of the distances between all occurrences ($o_1, o_2, \ldots, o_k$) pairs for a pattern P to calculate its rhythmic consistency:

$$v_{rhythm} = \left[\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} Dist\left(\bar{V}(o_a), \bar{V}(o_b)\right)\right] \Big/ \left[\frac{k(k-1)}{2}\right] \tag{12}$$

This value is a measure of how similar different occurrences are with respect to rhythm. Notice that two occurrences with the same notated rhythm presented at different tempi have a distance of 0. Consider the case where o_a has s times the tempo of o_b. In this case, $\bar{V}(o_b) = s\bar{V}(o_a)$, and $Dist(o_a, o_b) = \cos^{-1}(\frac{s[\bar{V}(o_a)\cdot\bar{V}(o_a)]}{s[\|\bar{V}(o_a)\|]^2}) = \cos^{-1} 1 = 0$. Occurrences

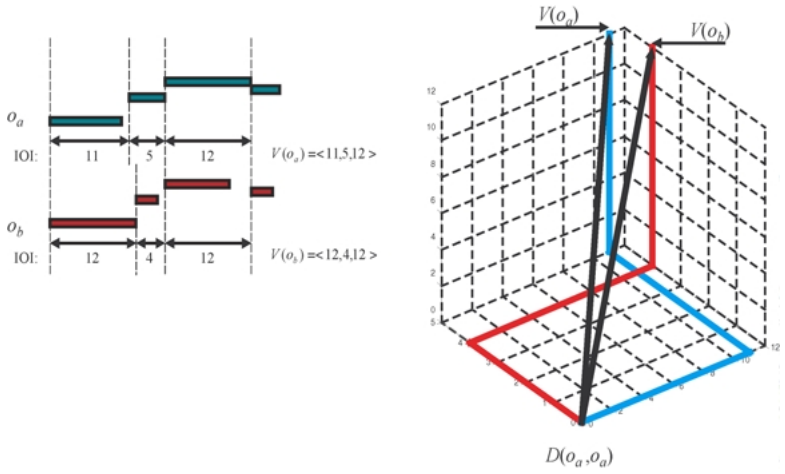

Figure 5. Rhythmic distance measure.

with similar rhythmic profiles have low distance, so this approach is robust with respect to performance and compositional variation. For instance, in the *Well-Tempered Clavier* (by J.S. Bach) often repeats fugue subjects at half speed. The rhythm vectors for the main subject statement and the subsequent stretched statement will thus have the same angle, and a distance of zero. Similarly, if two presentations of a theme have slightly different rhythmic inflections, their IOI vectors will nonetheless be quite similar.

3.10.3. Position. Noting that significant themes are often introduced near the start of a piece, we also characterize patterns according to the onset time of their first occurrence. Note that occurrences are sorted according to as patterns are identified, so the first occurrence is also the earliest occurrence:

$$v_{position} = \text{onset time of first event of first occurrence} \tag{13}$$

3.10.4. Register. Given the register values calculated for note events, the register value for a pattern P with occurrences $\{o_1, o_2, \ldots, o_k\}$, is equal to the average register of all events contained in those occurrences:

$$v_{register} = \frac{\sum_{i=1}^{k} \sum_{j=1}^{v_{length}+1} \text{Register of } j\text{th event of } i\text{th occurrence}}{k(v_{length}+1)} \tag{14}$$

3.11. Rating patterns

For each pattern, we have calculated several feature values. We are interested in comparing the importance of these patterns, and a convenient means of doing this is to calculate

percentile values for each parameter in each pattern, corresponding to the percentage of patterns over which a given pattern is considered "stronger" for a particular feature. These percentile values are stored in a feature vector:

$$F[P] = \langle p_{length}, p_{interval}, \ldots \rangle \tag{15}$$

We define "stronger" as either "less than" or "greater than" depending on the feature. Higher values are considered desirable for length, duration, interval counts, doublings and frequency; lower values are desirable for rhythmic consistency, pattern position and register.

The rating of a pattern P, given some weighting of features W, is:

$$Rating[P] \leftarrow W \cdot F[P] \tag{16}$$

3.12. Returning results

Patterns are then sorted according to their *Rating* field. This sorted list is scanned from the highest to the lowest rated pattern until some pre-specified number (k) of note events has been returned. Often, MME will rate a sub-sequence of an important theme highly, but not the actual theme, owing to the fact that parts of a theme are more faithfully repeated than others. As such, MME will return an occurrence of a pattern with an added margin on either end, corresponding to some ratio g of the occurrences duration, and some ratio of the number of note events h, whichever ratio yields the tightest bound.

In order to return a high number of patterns within k events, we use a greedy algorithm to choose occurrences of patterns when they are added: whichever occurrence adds the least number of events is used.

Output from MME is then a MIDI file consisting of a single channel of monophonic (single voice) note events, corresponding to important thematic material in the input piece.

4. Learning weights

We then wish to learn which value for W maximizes the performance of the system. This is related to the problem of learning value functions in preference-directed search (D'Ambrosio, 1994), where given a set of attributes, an ordering on value must be established. To give a simple example, when buying a computer, one might consider attributes such as price, speed, weight, memory, etc. Clearly, there are tradeoffs, so it is unlikely that one choice will be best in all attributes. Given enough purchase decisions, one might discover trends suggesting that one attribute is more important than another. Assigning weights to each attribute is a way of reflecting such a bias. Some potential weaknesses and assumptions should be noted with respect to this characterization:

1. We assume we have the correct attributes. Returning to the example of computer shopping, a consumer preference for a particular model might be that it comes in lime green and color may be a parameter considered by the model. Similarly, there are any number of additional clues an informed listener might pick up on with regards to thematic importance in music: stylistic context, knowledge of the structure, and expectations, among others.
2. We assume that the percentile representation is a useful way of characterizing the relative strength of a pattern with respect to a certain attribute. In this regard, we claim only that the percentile representation is a way of normalizing the attributes such that there is a uniform distribution across a fixed range.

For MME, we evaluate performance not based on consumer preference, but based on accepted musicological interpretations of the thematic content of musical pieces. Our reference of choice is Barlow's *A Dictionary of Musical Themes* (Barlow, 1975), which provides a set of "themes" for a large body of compositions. We enter the interval sequence associated with the "1st theme" of Barlow for every piece in our database. Future experiments may focus on the inclusion of secondary themes as well. We define the performance of a given weight vector W with respect to a given piece X, containing n patterns, as follows, where the ideal value is 1:

$$V(W, X) = \frac{n - \left(\begin{array}{c} \text{number of patterns with higher ratings than} \\ \text{the highest rated pattern containing the theme} \end{array} \right)}{n} \tag{17}$$

We note that the evaluation function is on the individual patterns rather than the output of the system. This allows the learning algorithm to take advantage of incremental improvements in performance, rather than the binary "include/not include" evaluation possible on the output. We do, however, use the occurrence margin in the evaluation function, to more consistently reflect the behavior of actual algorithm.

Given a collection of pieces $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$, the value of a particular weighting W is simply the average value across all pieces:

$$V(W, \mathbf{X}) = \frac{\sum_{i=1}^{n} V(W, X_i)}{n} \tag{18}$$

We employ a hill-climbing algorithm to learn W across a training set. It should be noted that given the equation for $Rating[P]$ (Eq. (16)), multiplying W by a scalar k increases all ratings commensurately with k. As such, $V(W, \mathbf{X}) = V(kW, \mathbf{X})$. To avoid redundancy, we consider choose to search across the space of W for which , or the surface of the unit sphere.

For hill-climbing, we choose successors ($\mathbf{W}_{new}$) for W along each axis:

$$W = \langle w_1, w_2, \ldots, w_n \rangle \tag{19}$$

$$\mathbf{W}_{new} = \bigcup_{i=1}^{n} \left(\begin{array}{c} \langle w_1, w_2, \ldots, w_i + \alpha_i^+, \ldots, w_n \rangle \cup \\ \langle w_1, w_2, \ldots, w_i - \alpha_i^-, \ldots, w_n \rangle \end{array} \right) \tag{20}$$

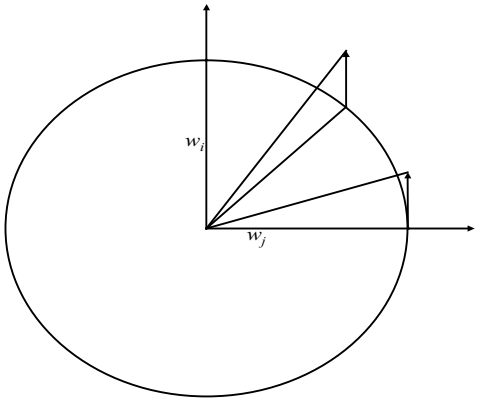

Figure 6. Updating W with fixed α_i value.

Each of the successors is then renormalized to the surface of the unit sphere. This is problematic, since if α_i is fixed, larger steps are taken in directions tangent to the surface (see figure 6). It might be better to take steps in terms of distance on the search surface, which is in turn equal to the angle between the original and successor weights. We choose α_i^+ and α_i^- to conform to a fixed angle (θ). The α values are then used to generate the successors in the hill-climbing search. See Eqs. (21) to (25) for the derivation of α from θ.

$$\cos(\theta) = \frac{W \cdot W_i}{\|W\| \, \|W_i\|} \quad \text{where} \quad \begin{array}{l} W_i \text{ is a successor along the } i\text{th axis, a distance of} \\ \theta \text{ radians from } W \end{array} \tag{21}$$

$$\cos(\theta) = \frac{\langle w_1, w_2, \ldots, w_n \rangle \cdot \langle w_1, w_2, \ldots, w_i + \alpha_i, \ldots, w_n \rangle}{\sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}\sqrt{w_1^2 + w_2^2 + \cdots + (w_i + \alpha_i)^2 + \cdots + w_n^2}} \tag{22}$$

$$\text{Let } A = w_1^2 + w_2^2 + \cdots + w_{i-1}^2 + w_{i+1}^2 + \cdots w_n^2 \tag{23}$$

$$\cos(\theta) = \frac{A + w_i(w_i + \alpha_i)}{\sqrt{A + w_i^2}\sqrt{A + (w_i + \alpha_i)^2}} \tag{24}$$

$$\alpha_i = \left[\begin{array}{l} \dfrac{-A\,(\cos(\theta))^2 w_i^2 + \cos(\theta)\sqrt{A + w_i^2}\sqrt{A\,w_i^2\left(A + w_i^2\right)(\sin(\theta))^2} + A\,w_i^2 - (\cos(\theta))^2 w_i^4 + w_i^4}{\left((\cos(\theta))^2 A + (\cos(\theta))^2 w_i^2 - w_i^2\right) w_i} \\ -\dfrac{A\,(\cos(\theta))^2 w_i^2 + \cos(\theta)\sqrt{A + w_i^2}\sqrt{A\,w_i^2\left(A + w_i^2\right)(\sin(\theta))^2} - A\,w_i^2 + (\cos(\theta))^2 w_i^4 - w_i^4}{\left((\cos(\theta))^2 A + (\cos(\theta))^2 w_i^2 - w_i^2\right) w_i} \end{array} \right] \tag{25}$$

We perform a random-restart hill-climbing search, randomly generating each element of the starting point from a uniform distribution in the range [0, 1] at each restart (see Algorithm 4).

Algorithm 4 Hill-climbing algorithm

Given pieces $\mathbf{X}$, where n is the number of weights in W:

for $i \leftarrow 1$ to the maximum number of restarts **do**
  $W \leftarrow$ random position in search space
  normalize(W)
  while true do
    $\{W_1, W_2, \ldots, W_{2n}\} \leftarrow$ **successors**(W)
    $\{maxValue, maxArg\} \leftarrow \max_{j=1}^{2n} V(\mathbf{X}, W_j)$
    if $maxValue > V(\mathbf{X}, W)$ **then**
      $W \leftarrow W_{maxArg}$
    else
      break
    end if
  end while
  if $V(\mathbf{X}, W) > V(\mathbf{X}, W_{best})$ **then**
    $W_{best} \leftarrow W$
  end if
end for

5. Results

A set of 60 pieces from the Baroque, Classical, Romantic, Impressionistic and 20th Century were used to train and test the software. Bach, Mozart, Beethoven, Brahms, Schubert, Mendelssohn, Dvorak, Smetana, Debussy, Bartok and Stravinsky are represented, in chamber, orchestral and solo piano works.

A few details of MME's configuration should be mentioned: the intervallic variety filter requires two distinct absolute intervals. Maximum pattern length is set to 12 transitions, and streams are broken where there are silent breaks longer than one and a half seconds. For the sake of result output and training, there is a margin of 0.5 on both ends for both events and duration. Up to 240 note events are returned for each piece, as compared with an average of over 8500 notes per piece originally. For the hill-climbing, we set $\theta = \frac{\pi}{50}$.

5.1. Preliminary results

Given even feature weighting, the primary theme was returned in 51 of the 60 pieces. Learning weights across this entire set, and testing across the same set, the primary theme was returned on 60 of the 60 pieces. These results are presented only to provide context for later results, and to provide some indication of the importance of learning appropriate weights.

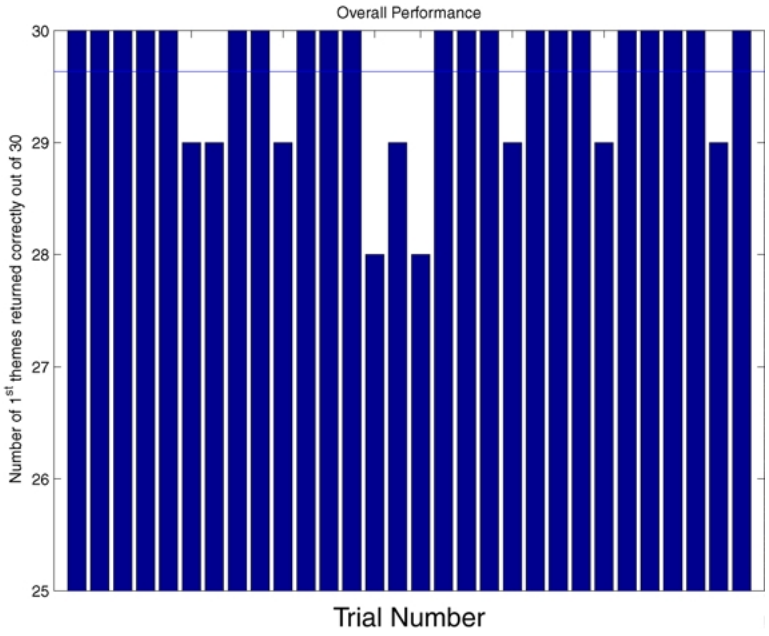

Figure 7. Trial results.

5.2. Training trials

We performed 30 trials, randomly selecting a 30-piece training set for each trial. During each trial, the hill-climbing algorithm was permitted 50 random restarts. These weights were then evaluated against the test set, consisting of the remaining 30 pieces. In two trials, MME identified 28 of the 30 primary themes, in seven trials 29 out of 30, and in 21 trials 30 out of 30, or on average roughly 29.6 out of 30, as compared with an expected average of 25.5 out of 30 using even weights (see figure 7).

5.2.1. Weights. Examining the weights learned during the trials, we get some idea of the relative importance of the different pattern features examined. The average and median weights across the 30 trials are listed in Table 2. Of particular interest is the negative weight for absolute interval count. Although our early experiments indicated that filtering patterns with low intervallic variety improves algorithm performance, it appears this parameter does not usefully distinguish the remaining patterns. The weight given to the "register" feature is perhaps most surprising, as we normally associate important "melodies" with the highest-sounding voice in a passage. "Position" is clearly the dominant feature, perhaps owing to our focus on primary themes, which tend to occur near the opening of pieces.

5.2.2. Errors. Three pieces were responsible for all errors in MME's output: the first movement of Mozart's *Symphony no. 40*, the second movement of Brahms' *Cello Sonata in*

Table 2. Feature weights.

Feature	Average weight	Median weight
Absolute interval count	−0.016249988	−0.021642894
Register	0.051727027	0.041694308
Doublings	0.085212347	0.055842776
Interval count	0.121993193	0.110731687
Frequency	0.119746216	0.125918866
Rhythmic consistency	0.176786867	0.181440092
Duration	0.233749767	0.237064805
Length	0.344768215	0.274449283
Position	0.819313306	0.872008477

E minor, and Brahms' *Academic Festival Overture*. In the first two cases, the proper theme was only partly returned in some trials, and in the last case, another theme sometimes dominated, albeit one that might be considered subjectively more prevalent than that listed first in Barlow.

Examining the Mozart example (see figure 8), the opening few notes exhibit a low absolute interval count (only minor seconds, $+/-1$), which explains why MME returned only the subsequent portion of the theme in some trials. This piece was included in 20 of the 30 test sets, and in three of those cases, the output was offset as described. In the remaining 17 cases, the proper theme was returned in full. In the case of the cello sonata, MME again selected only a portion of the 1st theme, in four of the 14 trials in which it appeared in the test set. This movement contains a great deal of repetition and variation, on the one hand offering a wealth of potentially important targets, and on the other, confusing the system due to its reliance on exact repetition.

The *Academic Festival Overture* contains a large number of themes, and in every trial, MME returned a fair number of them. The first theme listed in Barlow, however, was returned only six of the 10 times the piece appeared in the test set. In all cases, MME returned another theme (see figure 9).

5.2.3. Hill-climbing successor functions. Experiments were performed with various successor functions for the hill-climbing algorithm, to evaluate our approach. We also tested a fixed value, both renormalizing (to the surface sphere) and not renormalizing after steps were taken. For each of these three approaches, 50 random restarts were taken. We set α and θ equal to $\frac{\pi}{50}$. Using a fixed angle size, local maxima are discovered with fewer iterations,

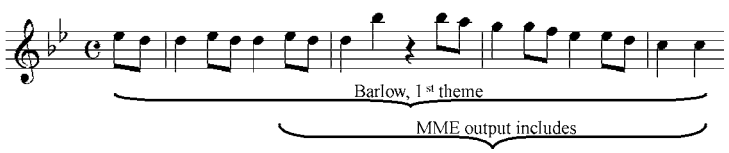

Figure 8. Mozart's *Symphone no. 40*, 1st theme.

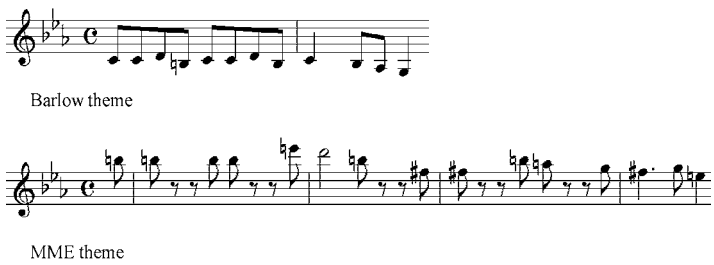

Figure 9. Themes from Brahms' *Academic Festival Overture*.

but the values of the learned weights are comparable (see Table 3). We contend that maxima are found more quickly using successors along the surface of the sphere because this surface corresponds to the actual search space. Other approaches converge more slowly as they take relatively small steps in directions not tangent to the surface.

5.2.4. Sample of output. MME's output from Smetana's *The Moldau* (a movement of *My Country*) is shown in figure 10. The first section A contains the 1st theme as indicated by Barlow. Section F contains a slight rhythmic variation on the same material, and section H presents the subsequent phrase. In addition, section B and D contain tonal variations of the same material (presented here in the major, whereas the main presentation is in the minor). To many listeners, these sections sound similar. This highlights a potential weakness of the algorithm: although the correct material is returned, there is redundancy in the output.

6. Summary

Identifying the major themes in a sophisticated musical work is a difficult task. The results show that MME correctly identifies the major themes in 100% of the test cases (when learning is employed), and identifies 85% of the major themes when learning is not used.

It is interesting to note that MME contains no deep musical knowledge, such as theory of melody, harmony, or rhythm. Rather, it works entirely from surface features, such as pitch contour, register, and relatively duration. We found, surprisingly, that register is not a good indicator of the thematic importance.

MME is computationally efficient. The system's overall complexity is dominated by the frequency calculation, which in the worst-case operates in $\Theta(N^3n^2)$ time, where M is the

Table 3. Comparison of successor functions.

Successor function	Mean/median number of iterations	Mean/median value of maxima
Normalizing, fixed α	37.98/38	0.999280221/0.999280221
Conventional, fixed α	36.12/35	0.99929344/0.999384179
Spherical, fixed θ	29.24/29	0.9993085/0.9993816

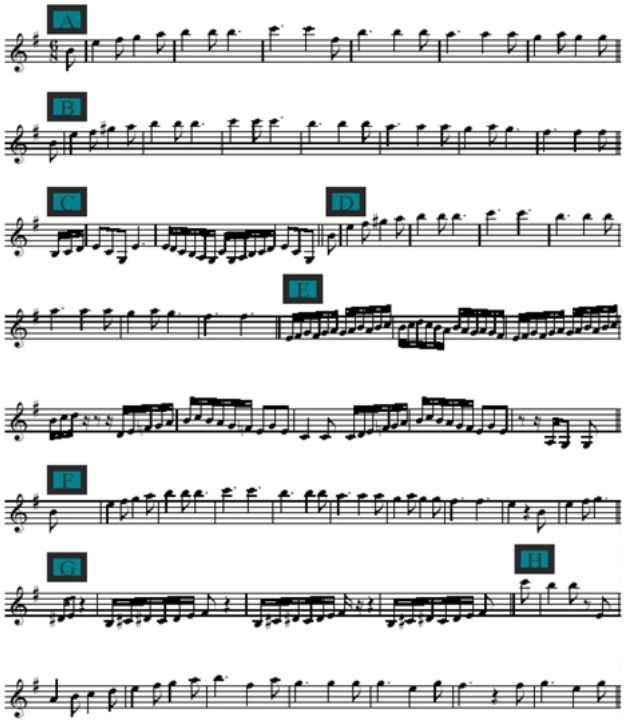

Figure 10. Output from Smetana's *Moldau*.

maximum pattern length under consideration, and n is the number of note events in the input piece. In practice however, we observe a sub-linear increase in running time with respect to piece size (see figure 11), and reasonable running times on even the largest input pieces.

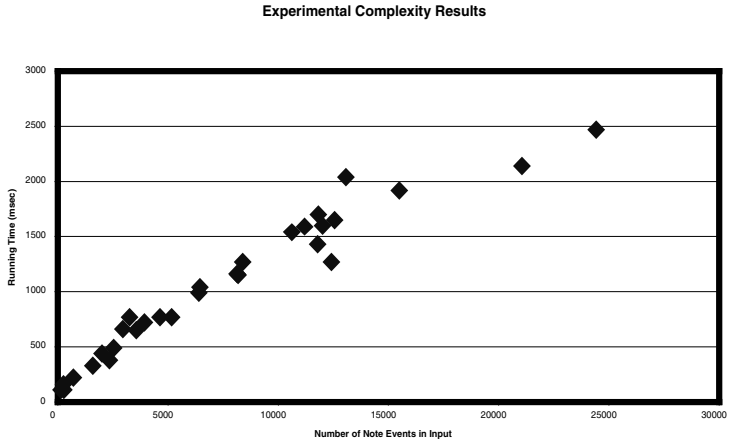

Figure 11. Experimental time complexity results (Pentium III 750MHz).

Acknowledgments

We gratefully acknowledge the support of the National Science Foundation under grant IIS-0085945, and The University of Michigan College of Engineering seed grant to the MusEn project. The opinions in this paper are solely those of the authors and do not necessarily reflect the opinions of the funding agencies. We also thank member of the MusEn research group for their comments. This group includes Greg Wakefield, Roger Dannenberg, Mary Simoni, Mark Bartsch and Bryan Pardo.

References

Alexandra and Uitdenbogerd. (1998). Manipulation of Music for Melody Matching. *ACM Multimedia, Electronic Proceedings*.

Ambrosio, J.G.D. and Birmingham, W. (1994). Preference-Directed Design. *AI EDAM*.

Bregman, A.S. (1990). *Scene Analysis: The Perceptual Organization of Sound*. The MIT Press.

Barlow, H. (1975). *A dictionary of Musical Themes*. New York: Crown Publishers.

Beek, P. van. (1996). The Design and Experimental Analysis of Algorithms for Temporal Reasoning. *Journal of Artificial Intelligence Research*.

Chai, W. (2001). Melody Retrieval on the Web. Masters Thesis, Massachusetts Institute of Technology.

Cope, D. (1996). *Experiments in Musical Intelligence*. A-R Editions.

Köchel, L.A.F. von et al. (1850–1964). *Chronological-Thematic Catalog of the Complete Works of Wolfgang Amadé Mozart*. Various editions.

Lu, L., You, H., and Zhang, H. (2001). A New Approach to Query by Humming in Music Retrieval. *Proceedings of ICME*.

Rivest, R.L. et al. (1999). *Introduction to Algorithms*. The MIT Press.

Simoni, M. et al. (2000). A Theoretical Framework for Electro-Acoustic Music. In *Proceedings of ICMC*.

Temperley, D. (1999). Modeling Meter and Harmony: A Preference-Rule Approach. *Computer Music Journal*, 15(1), 10–27.

Temperley, D. (2000). A Model for Contrapontal Analysis. Unpublished.

Tseng, Y.-H. (1999). Content-Based Retrieval for Music Collections. In *Proceedings of SIGIR*.