Thematic Extractor

Colin Meek
University of Michigan
1101 Beal Avenue
Ann Arbor MI 48104
1-734-763-1561
meek@umich.edu

William P. Birmingham
University of Michigan
1101 Beal Avenue
Ann Arbor MI 48104
1-734-936-1590
wpb@umich.edu

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.

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 [1]) to catalog the works of a composer or performer. Moreover, musicians often use thematic indices (e.g., Barlow's *A Dictionary of Musical Themes* [2]) 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.

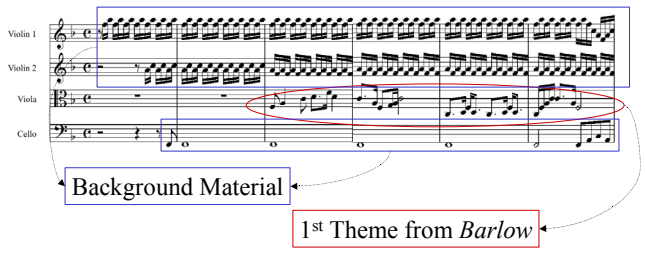


Figure 1: Sample Thematic Extraction from opening of Dvorak's *American Quartet*

Theme extraction using computers has proven very difficult. The best known methods require some 'hand tweaking' [3] to at least provide clues about what a theme may be, or generate thematic listings based solely on repetition and string length [4]. 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 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.
- 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 into note sequences and seeing how often sequences repeat, we identify the themes. Breaking up involves examining all note sequence lengths of 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.

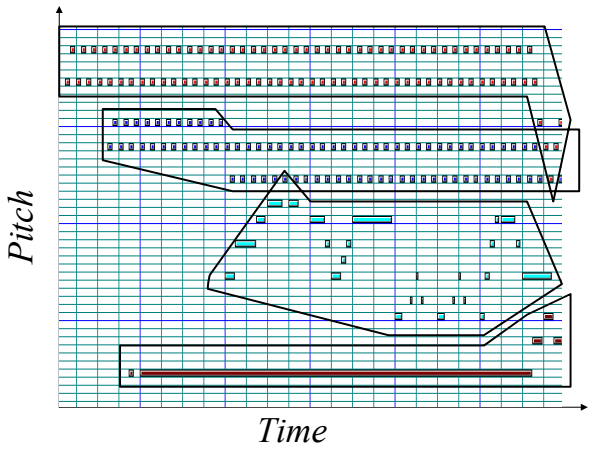


Figure 2: Opening Phrase of Dvorak's "*American*" Quartet

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.

1.1 Problem Formulation

Input to MME is a set of note events making up a musical composition $N = \{n_1, n_2 ... n_3\}$. 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 = <*onset*, *offset*, *pitch*>. We note that several other valid representations of a musical composition exist, taking into account amplitude, timbre, meter and expression markings among others [6]. 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 t published thematic catalogs represent common convention. These catalogs thereby provide a concrete measure by which the system can be evaluated..

2. 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:

1. Input
2. Register
3. Stream segregation
4. Filter top voice
5. Calculate event transitions
6. Generate event keys
7. Identify and filter patterns
8. Frequency
9. Compute other pattern features
10. Rate patterns
11. Return results

2.1 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.

2.2 Register

Register is an important indicator of perceptual prominence [10]: 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.

```
Given the input set of events N[]:
 1.  Sort(N, onset[N])
 2.  ActiveList ← NULL
 3.  index ← 0
 4.  while index < n
 4.     onset ← Onset[N[index]]
 5.     • remove all inactive events
 6.     Remove(ActiveList, Offset[N] • Onset)
 7.     • add all events with the same onset
 8.     while index < n – 1 and Onset[N[index]] = onset
 9.        Register[N[index]] ← 0
10.           add N[index] to ActiveList
11.           increment index
12.        • update Register value of active events
13.        Sort(ActiveList, Pitch[ActiveList])
14.        n ← Size[ActiveList] – 1
15.        for j ← 0 to n
16.           register ← n - j / n
17.           if register > Register[ActiveList[j]]
18.              Register[ActiveList[j]] ← register
```

Algorithm 1: Calculating Register

Table 1: Register values at each iteration of register algorithm

Adding	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	ActiveList
e_0	0								$\{e_0\}$
e_1	1	0							$\{e_0, e_1\}$
e_2	1	0	1/2						$\{e_0, e_1, e_2\}$
e_3	1	0	1	0					$\{e_2, e_3\}$
e_4, e_5	1	0	1	2/3	1/3	0			$\{e_2, e_3, e_4, e_5\}$
e_6, e_7	1	0	1	2/3	1/3	0	1/2	1	$\{e_4, e_6, e_7\}$

We need to define the notion of concurrency more precisely. Two events with intervals $I_1 = [s_1,e_1]$ and $I_2 = [s_2,e_2]$ are considered concurrent if there exists an common interval $I_c = [s_c,e_c]$ such that $s_c < e_c$ and $I_c \subseteq I_1 \wedge I_c \subseteq I_2$. The simplest way of computing these values is to walk through the event set ordered on onset time, maintaining a list of (notes that are on) events, or events sharing a common interval (see Algorithm 1).

Consider the example piece in Figure 3. The register value assigned to each event $\{e_0 \ldots e_7\}$ at each iteration is shown in Table 1.

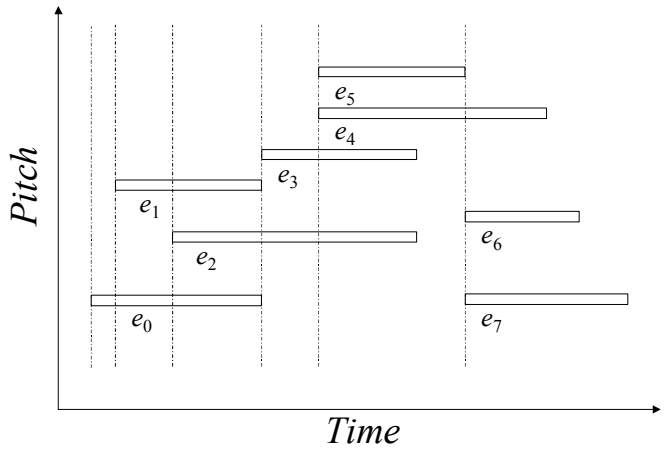


Figure 3: Register, Example Piece

2.3 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 2 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 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 [7], perform stream segregation on abstracted music, i.e., note-event representation, they have trouble with overlapping voices [8], as seen between the middle voices in Figure 2.

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 (see Algorithm 1), wherein events are removed from the active list for their particular channel some ratio (0.5) of their duration from their onset, and as such avoid being falsely labeled as "lower-sounding" notes. For instance, an event in the time interval [30, 50] will be removed from the active list when the sweep reaches time 40.

Additionally, when long pauses (greater than some time constant) are found in a stream, the stream is broken at that point. 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 indicate events using the notation $e_{stream,\ index}$, such that $e_{0,1}$ indicates the second note of the first stream.

2.4 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: $e_s = \{<0, 1, 60>, <1, 2, 62>, <2, 3, 64>, <3, 4, 62>, <4, 5, 60>\}$ has contour $c_s = \{+2, +2, -2, -2\}$.

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 + 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]:

$$real\_interval_{s,i} = Pitch[e_{s,i+1}] - Pitch[e_{s,i}]$$

$$c_{s,i} = \begin{cases} real\_interval_{s,i}, & \text{if } -12 \le real\_interval_{s,i} \le +12 \\ -\bmod_{12} -real\_interval_{s,i}, & \text{if } real\_interval_{s,i} < -12 \\ \bmod_{12} real\_interval_{s,i} & \text{o.w.} \end{cases}$$

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 above example, the IOI values are simply {1, 1, 1, 1}:

$$IOI[e_{s,i}] = Onset[e_{s,i+1}] - Onset[e_{s,i}]$$

2.5 Calculating Keys

To efficiently uncover patterns, or repeating sequences, we assign a key k to each event in the piece that uniquely identifies a sequence of m intervals, where m is the maximum pattern length under consideration. Length refers to the number of intervals in a pattern, one less than the number of events. The keys must exhibit the following property:

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

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]:

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

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

Equation 1

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

Equation 2

$$k_{s,i}(n) = \begin{cases} 26 * k_{p,i}(n-1) + k_{p,i+n-1}(1) & \text{if } n \le |c_s| - i \\ k_{s,i}(|c_s| - i) * 26^{n-|c_s|+i} & o.w. \end{cases}$$

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 $i \ge |c_s|$, $k_{s,i}(1) = 0$, 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 $k_{s,i+1}(n-1)$:

Equation 3

$$k_{s,i+1}(n-1) = k_{s,i}(n) - k_{s,i}(1) * 26^{n-1}$$

This in turn allows us to calculate the subsequent key value in constant time, using Equation 2.

Using Equation 1 and Equation 2, we can calculate the key if the first event in a stream 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* [9]). Equation 3 allows us to calculate the key of each subsequent event in constant time (as with the *Rabin-Karp* algorithm [9]). 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_0 = {-1, 0, +1, -1, 0, +1, -1, 0 +8}.

First we calculate the key value for the first event ($k_{0,0}(4)$), using Equation 1 and Equation 2 recursively:

$$\begin{aligned} k_{0,0}(4) &= 26 * k_{0,0}(3) + k_{0,3}(1) \\ &= 26 * (26 * k_{0,0}(2) + k_{0,2}(1)) + 12 \\ &= 26 * (26 * (26 * k_{0,0}(1) + k_{0,1}(1)) + 14) + 12 \\ &= 26 * (26 * (26 * 12 + 13) + 14) + 12 \\ &= 220076 \end{aligned}$$

Then we calculate the remaining key values:

$$k_{0,1}(3) = k_{0,0}(4) - k_{0,0}(1) * 26^3 = 9164 \quad \text{(Equation 3)}$$

$$k_{0,1}(4) = 26 * k_{0,1}(3) + k_{0,4}(1) = 238277 \quad \text{(Equation 2)}$$

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

$k_{0,2}(4) = 254528$ $k_{0,3}(4) = 220076$ $k_{0,4}(4) = 238277$ $k_{0,5}(4) = 254535$
$k_{0,6}(4) = 220246$ $k_{0,7}(4) = 242684$ $k_{0,8}(4) = 369096$ $k_{0,9}(4) = 0$

2.6 Identifying and Filtering Patterns

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

Equation 4

$$\forall n, 0 < n \le m : k_{s,i}(n) = \left\lfloor \frac{k_{s,i}(m)}{26^{m-n}} \right\rfloor$$

Events are then sorted on key so that pattern occurrences are adjacent in the ordering. We make a pass through the list for pattern lengths from $n = [m \ldots 2]$, 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 $k(n)$ - calculated using Equation 4 – 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.

Continuing with the Mozart example, sorting the keys we get: $\{k_{0,9}, k_{0,0}, k_{0,3}, k_{0,6}, k_{0,1}, k_{0,4}, k_{0,7}, k_{0,2}, k_{0,5}, k_{0,8}\}$.

On our first pass through the list, for $n = 4$, we identify patterns $\{k_{0,0}, k_{0,3}\}$ and $\{k_{0,1}, k_{0,4}\}$, since there keys are identical. During the second pass, for $n = 3$, we identify patterns $\{k_{0,0}, k_{0,3}\}$, $\{k_{0,1}, k_{0,4}\}$ and $\{k_{0,2}, k_{0,5}\}$, noting that $k_{0,2}/26^{4-3} = k_{0,5}/26^{4-3}$ (which by Equation 4 indicates that a pattern of length three exists.) Similarly, we identify the following patterns for $n = 2$: $\{k_{0,0}, k_{0,3}, k_{0,6}\}$, $\{k_{0,1}, k_{0,4}\}$ and $\{k_{0,2}, k_{0,5}\}$. The patterns are shown in Table 2.

Table 2: Patterns in opening phrase of Mozart's *Symphony no. 40*

Pattern	Occurrences at	Characteristic interval sequence
P_0	$e_{0,0}, e_{0,3}$	{-1, 0, +1, -1}
P_1	$e_{0,1}, e_{0,4}$	{0, +1, -1, 0}
P_2	$e_{0,0}, e_{0,3}$	{-1, 0, +1}
P_3	$e_{0,1}, e_{0,4}$	{0, +1, -1}
P_4	$e_{0,2}, e_{0,5}$	{+1, -1, 0}
P_5	$e_{0,0}, e_{0,3}, e_{0,6}$	{-1, 0}
P_6	$e_{0,1}, e_{0,4}$	{0, +1}
P_7	$e_{0,2}, e_{0,5}$	{+1, -1}

We associate a vector of parameter values $V_i = < v_1, v_2, \ldots, v_n >$ and a set of occurrences to each pattern. 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 zero-padding. Two other filtering criteria are considered as well: intervallic variety, and doublings.

2.6.1 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 $+n$ or $-n$ 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.

2.6.2 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 o_a and o_b, with initial events $e_{s1,i1}$ and $e_{s2,i2}$ respectively, and length n, are considered simultaneous if and only if $\forall j, 0 \le j \le n : e_{s1,i1+j}$ 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 [11] -- $e_{s1,i1}$ before (*b*) $e_{s2,i2}$ or the inverse (*bi*) (see Algorithm 2):

We check each occurrence of a pattern against every other occurrence. Note that since occurrences are sorted on onset, we know that if o_i and o_j are not doublings, where $j > i$, o_i cannot double o_k for all $k > j$. This provides a way of curtailing searches for doublings in our algorithm, and provides significant performance gains (experimentally, a tenfold improvement). This is because partial doublings rarely occur, where only some subset of corresponding intervals is simultaneous.

```
Given a pattern P with n occurrences in O[] and length l
 1.  for i ← 0 to n − 2
 2.     for j ← i + 1 to n − 1
 3.        if ~Remove[O[i]] and ~Remove[O[j]]
 4.           Simultaneous = true
 5.           for k ← 0 to l
 6.              if ~Intersects(e_Stream[O[i]],Index[O[i]], e_Stream[O[j]],Index[O[j]]) then
 7.                 Simultaneous ← false
 8.                 k ← l + 1
 9.           if Simultaneous then
10.              if Pitch(e_Stream[O[i]],Index[O[i]]) > Pitch(e_Stream[O[j]],Index[O[j]])
11.                 Remove[j] ← true
12.                 Doubled[i] ← true
13.              else
14.                 Remove[i] ← true
15.                 Doubled[j] ← true
16.           else
17.              j ← n
18.  Remove(O, Remove[O])
```

Algorithm 2: Filter Doublings

This doubling filtering occurs before other computations, and thus influences frequency. We, however, retain the doubling information (Lines 12 and 15, Algorithm 2), 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 removed from consideration. Doublings serve to reinforce a voice, and as such do not constitute repetition.

2.7 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 $\{C_0, C_1, ..., C_{v_{length}-1}\}$, the frequency of occurrences is defined as follows:

$$\frac{\sum_{l=v_{length}}^{2} \sum_{j=0}^{v_{length}-1} \text{non - redundant and un - filtered occurrences of } \{C_j, C_{j+1}, \ldots, C_{j+l-1}\} * l}{v_{length}}$$

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_0: $c_0 = \{-2,+2,-2,+2,-5,+5,-2,+2,-2,+2,-5,+5,-2,+2,-2,+2\}$, and the pattern $\{-2,+2,-2,+2,-5\}$. Clearly, there are two complete occurrences at $e_{0,0}$ and $e_{0,6}$, but also a partial occurrence of length four at $e_{0,12}$. The frequency is then 2.8 for this pattern.

To efficiently calculate frequency, we first construct a set of pattern occurrence lattices, on the following binary occurrence relation $\prec$:

Given occurrences o_1 and o_2 characterized by event sequences E_1 and E_2, $o_1 \prec o_2 \Leftrightarrow E_1 \subset E_2$. In other words, each occurrence in the lattice covers all patterns occurrences containing a subsequence of that occurrence.

As such, in establishing frequency, we need consider only those patterns covered by occurrences of P 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 $n > 2$, there are at most two occurrences of length $n - 1$, 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 $n - 1$ will be covered by the occurrence in these instances, so we need only generate links to occurrences of length $n - 1$ in the lattices. The branching factor is thus limited to two.

The lattice is described as follows: given a node representing an occurrence of a pattern o with length l, the left child is an occurrence of length $l - 1$ beginning at the same event. The right child is an occurrence of length $l - 1$ beginning at the following event. The left parent is an occurrence of length $l + 1$ beginning at the previous event, and the right parent is an occurrence of length $l + 1$ beginning at the same event. Consider the patterns the Mozart excerpt (see Table 2): P_0's first occurrence, with length 4 and at $e_{0,0}$, directly covers two other occurrences of length 3: P_2's first occurrence at $e_{0,0}$ (left child) and P_3's first occurrence at $e_{0,1}$ (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. Each note event in the piece contains a pointer to an occurrence, such that as occurrences are added, lattice links can be built in constant time (see Algorithm 3).

Consider the patterns identified in the Mozart example (Table 2), from which we build the lattice in Figure 1. When the first occurrence of pattern P_4 is inserted, *o_left* = the first occurrence of P_3, and *o_right* = null. Since P_3 has the same length as P_4, we check the right parent of the *o_right*, and update the link between those occurrences of P_1 and P_4. Other links are updated in a more straightforward manner.

```
Given a series of n patterns P[]
 1.  for i ← 0 to n − 1
 2.    O ← Occurrences[P[i]]
 3.    for j ← 0 to Size[O]
 4.      • occurrence pointed to by the first event of O
 5.      o_right ← Occurrence[e_Stream[O[j]], e_Index[O[j]]]
 6.      • occurrence pointed to by the preceding event
 7.      if Index[O[j]] = 0
 8.        o_left ← null
 9.      else
10.        o_left ← Occurrence[e_Stream[O[j]], e_Index[O[j]]-1]
11.      • we consider three cases for the value of o_left
12.      if o_left = null
           • we learn nothing about the lattice
13.      else if Length[o_left] > Length[O[j]]
14.        Right_Child[o_left] ← O[j]
15.      else
16.        Right_Child[Right_Parent[o_left]] ← O[j]
17.      • we consider two cases for the value of o_right
18.      if o_right = null
           • we learn nothing about the lattice
19.      else
20.        Right_Parent[O[j]] ← o_right • used in line 16
21.        Left_Child[o_right] ← O[j]
22.        Occurrence[e_Stream[O[j]], e_Index[O[j]]] ← O[j]
```

Algorithm 3: Lattice Construction

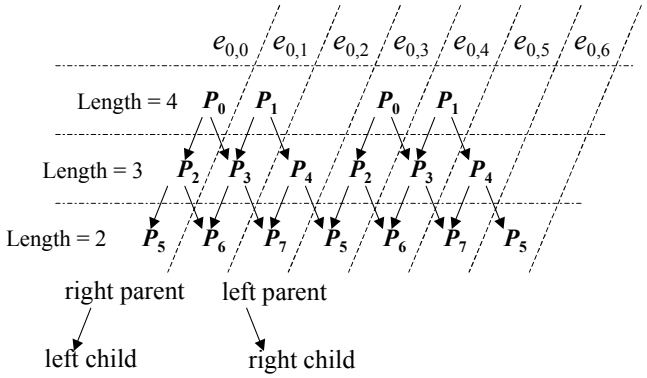


Figure 4: Lattice for the First Phrase of Mozart's *Symphony no. 40*

From this 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 4).

Take for instance pattern P_2 in the Mozart example. By breadth-first traversal, starting from either occurrence of P_2, the following elements are added to Q: P_2, P_5 and P_6. First, we add the two occurrences of P_2, tagging events $e_{0,0}$, $e_{0,1}$, … , $e_{0,5}$, and setting $v_{frequency} \leftarrow 6$. The first two occurrences of P_5 contain tagged events, so we reject them, but the third occurrence at $e_{0,6}$ is un-tagged, so we tag $e_{0,6}$, $e_{0,7}$, $e_{0,8}$ and set $v_{frequency} \leftarrow 6 + 2$. All occurrences of P_6 are tagged, so the frequency of P_2 is equal to 8 / 3.

```
Given a pattern P:
 1.  id ← unique identifier for pattern
 2.  Tag[P] ← id
 3.  push(Q, P)
 4.  while ~empty(Q)
 5.    • add chbildren to Queue (DFS)
 6.    pop(Q, p)
 7.    o_left ← Left_Child[Occurrences[p][0]]
 8.    o_right ← Right_Child[Occurrences[p][0]]
 9.    if o_left ~= null and Tag[Pattern][o_left] ~= id
10.      Tag[Pattern[o_left]] ← id
11.      push(Q, Pattern[o_left]])
12.    if o_right ~= null and Tag[Pattern][o_right] ~= id
13.      Tag[Pattern[o_right]] ← id
14.      push(Q, Pattern[o_left]])
15.    • count non-redundant occurrences of p
16.    for i ← 0 to Size[Occurrences[p]] − 1
17.      if events in Occurrences[p][i] have Tag ~= id [1]
18.        set Tag ←
             id for all events in Occurrences[p][i]
19.        v_frequency[P] ← v_frequency[P] + Length[Occurrences[p][i]]
20.        v_doublings[P] ← v_doublings[P] + Length[Occurrences[p][i]]
21.  v_frequency[P] ← v_frequency[P] / v_length[P]
22.  v_doublings[P] ← v_doublings[P] / v_length[P]
```

Algorithm 4: Calculating Frequency

2.8 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, $v_{register}$).

2.8.1 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 $D(o) = \mathit{Offset}[e_{s2,i2}] - \mathit{Onset}[e_{s1,i1}]$. For a pattern P, with occurrences o_0, o_1, … , o_{n-1}, the distance parameter is calculated to be the average duration of all occurrences:

$$v_{duration} = \frac{\sum_{i=0}^{n-1} D(o_i)}{n}$$

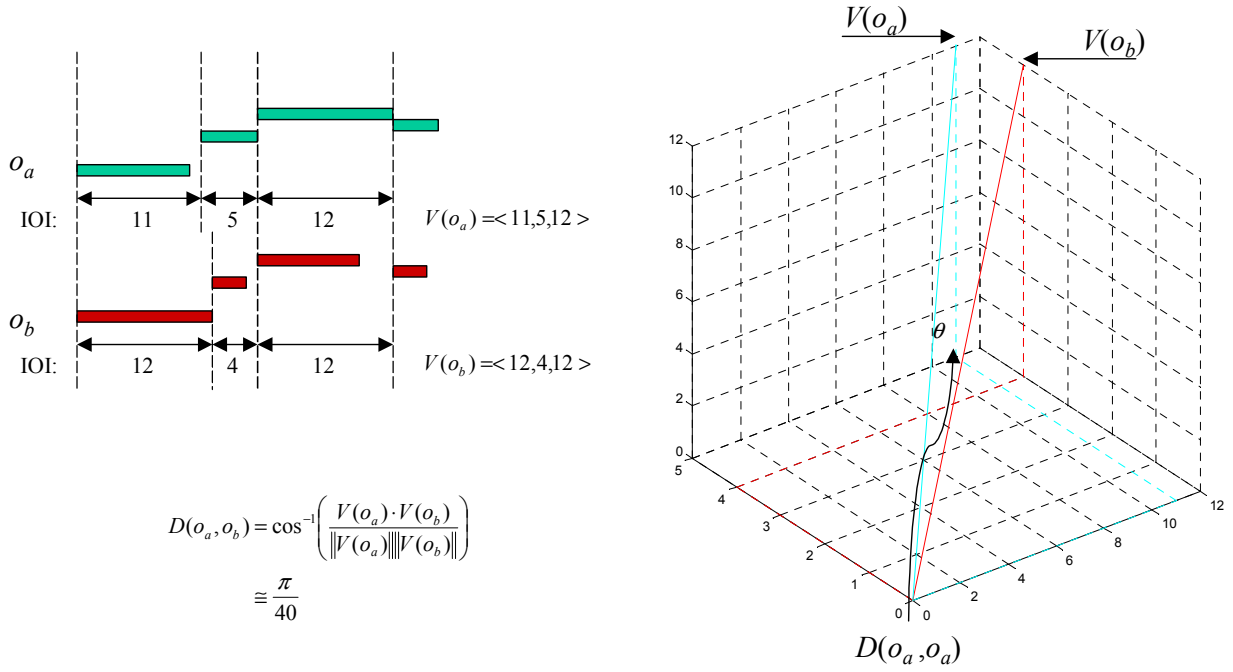
2.8.2 Rhythmic Consistency

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. For occurrence o, with events E_0, E_1, … , $E_{length\ -\ 1}$,[2] the IOI vector is $V(o) = < IOI[E_0], IOI[E_1], ..., IOI[E_{length-1}] >$. The rhythmic distance between a pair of occurrences o_a and o_b is then the angle distance between the vectors $V(o_a)$ and $V(o_b)$:

$$D(o_a, o_b) = \cos^{-1}\left(\frac{V(o_a) \cdot V(o_b)}{\|V(o_a)\| \|V(o_b)\|}\right)$$

[1] If the first and last events of *Occurrences*[*p*][*i*] are un-tagged, then we can assume the occurrence has not been counted even in part, since previously considered occurrences are necessarily of greater or equal length. As such, only the first and last events are examined here.

[2] We use the notation E_j to refer to an arbitrary event $e_{s,i}$. Note that E_j and E_{j+1} refer to consecutive events $e_{s,i}$ and $e_{s,i+1}$.



$$D(o_a, o_b) = \cos^{-1}\left(\frac{V(o_a)\cdot V(o_b)}{\|V(o_a)\|\|V(o_b)\|}\right) \cong \frac{\pi}{40}$$

Figure 5: Rhythmic Distance Measure

A 3-dimensional example of the rhythmic distance calculation between two occurrences o_a and o_b is shown in

Figure 5.

We take the average of the distances between all occurrence (o_0, o_1, ... , o_{n-1}) pairs for a pattern P to calculate its rhythmic consistency:

$$v_{rhythm} = \frac{\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} D(V(o_i), V(o_j))}{\frac{n(n-1)}{2}}$$

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 k times the tempo of o_b. In this case, $V(o_b) = kV(o_a)$, and $D(V(o_a), V(o_b)) = D(V(o_a), kV(o_a)) = 0$.

Occurrences 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* 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.

2.8.3 Position

Noting that significant themes are sometimes introduced near the start of a piece, we also characterize patterns according to the onset time of their first occurrence (o). Note that occurrences are sorted according to *Onset* as patterns are identified, so the first occurrence is also the earliest occurrence:

$$v_{position} = Onset[e_{Stream[o],Index[o]}]$$

2.8.4 Register

Given the register values calculated for note events, the register value for a pattern P with occurrences o_0, o_1, ... , o_{n-1}, is equal to the average register of all events contained in those occurrences:

$$v_{register} = \frac{\sum_{i=0}^{n-1}\sum_{j=0}^{v_{length}} Register[e_{Phrase[o_i],Index[o_i]+j}]}{n*(v_{length}+1)}$$

2.9 Rating Patterns

For each pattern P, 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:

$$F[P] = < p_{Length}, p_{interval\_count}, \ldots, p_{register} >$$

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:

$$Rating[P] \leftarrow W \cdot F[P]$$

2.10 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 are 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 a MIDI file consisting of a single channel of monophonic (single voice) note events, corresponding to important thematic material in the input piece.

3. 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 required a minimum of at least zero distinct intervals, and two distinct absolute intervals. Maximum pattern length is set to 12 transitions, and streams are broken with silences 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. We employ a hill-climbing algorithm to discover good values for *W*.

3.1 Preliminary Results

Given even feature weighting, the primary theme was returned in 51 of the 60 pieces. Learning weights *W* a*cross* 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.

3.2 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 6.)

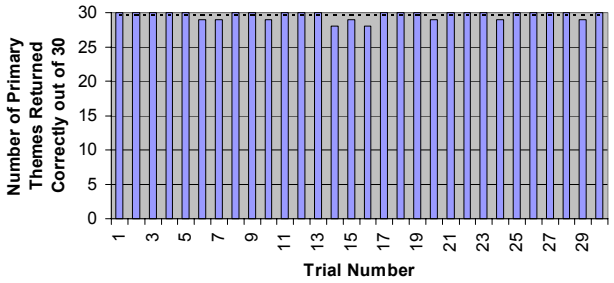


Figure 6: Trial Results

3.2.1 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 3.

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.

Table 3: 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

3.2.2 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 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 7), 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.

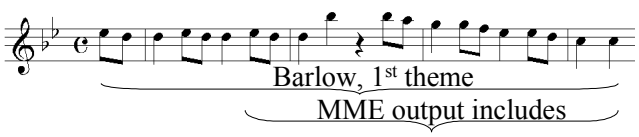


Figure 7: Mozart *Symphony no. 40* 1st Theme

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 8).

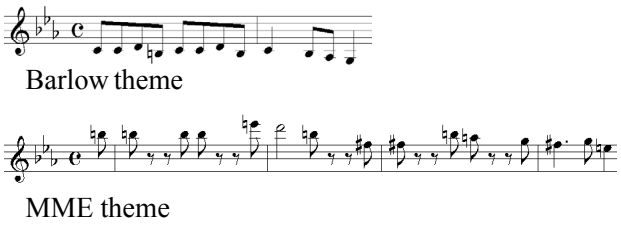


Figure 8: Themes from Brahms' *Academic Festival Orchestra*

3.2.3 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.

3.2.4 Popular Music

MME has been tested on several pieces of popular music, though we present no formal results in the absence of an accepted benchmark for system performance in this genre. Across 20 songs, ranging from the Beatles to Nirvana, an untrained version of MME returned the chorus where applicable, and what we considered to be significant "hooks" in all cases.

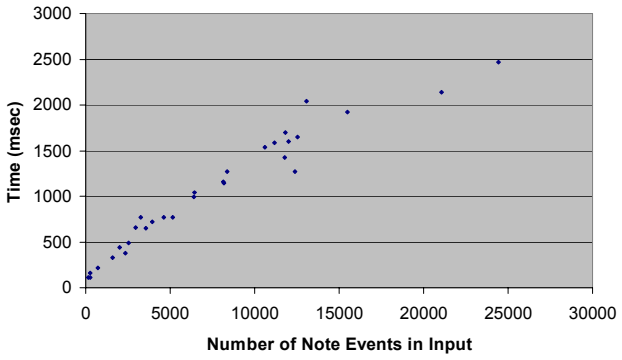


Figure 9

4. 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 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 $\Theta(m^3 n^2)$ time, where m is the maximum pattern length under consideration, and n is the number of note events in the input piece. In practice, however, we observe sub-linear performance (see Figure 9), and reasonable running times on even the largest input pieces.

5. Acknowledgements

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.

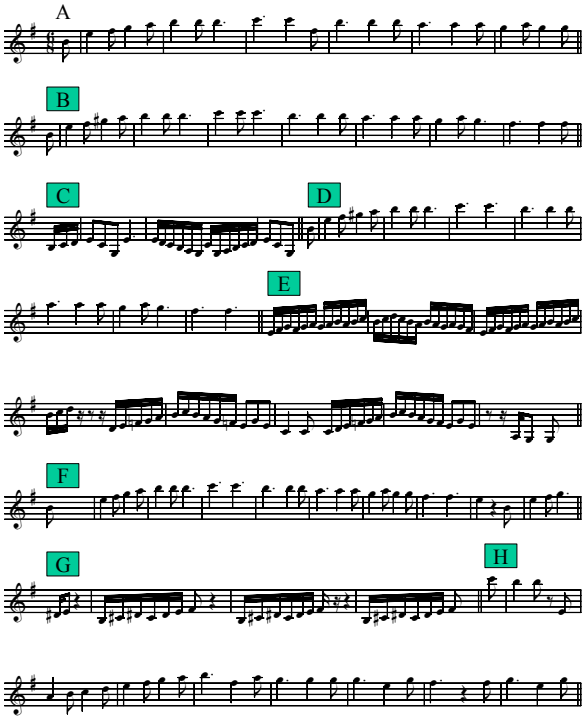


Figure 10: Output from Smetana's *Moldau*

6. References

[1] Ludwig Ritter von Köchel. *Chronologisch-thematisches Verzeichnis sämtlicher Tonwerke Wolfgang Amadé Mozarts; nebst Angabe der verlorengegangenen, ange l'angene, von fremder Hand bearbelteten, zwelfelhaften und unterschobe-nen Kompositionen*. Wiesbaden, Breitkopf & Härtel, 6th edition, 1964.

[2] H. Barlow. *A dictionary of Musical Theme*s. Crown Publish-ers, New York, 1975.

[3] David Cope. *Experiments in Musical Intelligenc*e. A-R Editions, 1996.

[4] Alexandra and Uitdenbogerd. Manipulation of music for melody matching. *ACM Multimedia, Electronic Proceedings*, 1998.

[5] Y.-H. Tseng. Content-based retrieval for music collections. *SIGIR*, 1999.

[6] M. Simoni, C. Rozell, C. Meek, and G. Wakefield. A theoretical framework for electro-acoustic music. *ICMC*, 2000.

[7] David Temperley. Modeling meter and harmony: A preference-rule approach. *Computer Music Journal*, 23.

[8] David Temperley. A model for contrapontal analysis. *unpublished*.

[9] R. L. Rivest T. H. Cormen, C. E. Leiserson. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1999.

[10] A.S. Bregman. *Auditory Scene Analysis: The Perceptual Organization of Sound*. MIT Press.

[11] Peter van Beek. The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research*, 1996.

[12] Joseph G. D'Ambrosio, William O, Birmingham. Preference-Directed Design. *AI EDAM*, 1994. R. L. Rivest T. H. Cormen, C. E. Leiserson. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1999.