Fugue Analysis with Hidden Markov Models

by Jonathan Crosmer

May 6, 2009

CSCE 970: Pattern Recognition

with Dr. Stephen Scott

Special thanks to Dr. Stanley Kleppinger

Abstract

We propose and implement an algorithm for the analysis of fugues and other polyphonic music. The analysis takes a set of one or more themes as input and generates a representative hidden Markov model for each, using prior domain knowledge to model likely transformations. We demonstrate the analysis with a GUI and provide evaluation by an independent expert. The output is musically sensible, and the model captures some sophisticated aural relationships. Our model can provide a starting point for human analysis and could serve as an educational tool.

Introduction

Musical analysis requires the ability to recognize musical themes as they are transformed and manipulated throughout a piece. In a contrapuntal work, such as a fugue or invention, the first theme is called the "subject," and the second is the "countersubject." A music theorist can identify not only exact repetitions of the subject, but also transformations (such as modal alteration or the "tonal answer") and fragments of subject material. A fugue typically has several complete subject statements and some sections that have only fragments. The process of identifying passages that relate to the subject is challenging, as material may be manipulated in creative ways, but it is also quantitative and suggests automation. We apply hidden Markov models (HMMs) to this problem and create a tool for musical analysis. Our tool focuses on the problem of recognizing aural relationships within a piece. The HMMs used are not suitable for generating music, as they operate on a low level, ignoring more abstract concepts like contour and form. However, the tool may be useful for music theorists or students interested in quickly performing a preliminary analysis of the thematic manipulations in a contrapuntal work.

Problem

Assume we can represent a musical score as a set of independent voices, where each voice is a long sequence of notes spanning the length of the piece. We will define "note" more carefully later, when we describe our method; for now, the intuitive musical definition will suffice. A "theme" can be any sequence of notes that we desire to match. Usually, the voices enter one at a time. When this is the case, the subject is the subsequence of notes in the first voice between the beginning and the entrance of the second voice; the countersubject (if there is one) is the subsequence in the same voice between the entrances of the second and third voices. We assume that the initial statement of each theme is identified in advance.

Given a set of note sequences and one or more "themes," the problem is to identify subsequences that are likely to be derivatives of themes or theme fragments. This problem is well-studied by music theorists and is commonly proposed to students of counterpoint on examinations. Solving this problem is often the first step in analyzing a contrapuntal piece.

Related Work

HMMs have been applied to other kinds of music analysis. Chai and Vercoe [2001] used HMMs to group melodies by similarity. They tried several encoding methods to represent the melodies as symbol strings, typically by focusing on a single musical element (rhythm, pitch, or contour). By using simple models and no musical assumptions beyond those needed for encoding, they limit the bias introduced into the models. However, this also limits the analytical power to simple classification.

Lubiw and Tanur [2004] attack a problem very similar to ours, but with geometric approach emphasizing low algorithmic complexity. Their algorithm searches the score for

patterns that are similar to a given theme, applying a similarity measure that penalizes interval substitutions. It does allow for insertions and deletions in pattern matches, but it does not appear to be designed or suited for a general fragment search (that is, their primary goal was to find matches for the entire theme). They also state that changes to the rhythm of a theme are not easily detected by their algorithm. One advantage to their approach is that it can handle polyphonic music without requiring a prior reduction to monophonic voices. Since we use HMMs, the input must be separated by voice so that each voice can be encoded as a symbol string.

Radicioni and Botta [2006] use HMMs to analyze pieces without being given any themes to match. Instead, their algorithm predicts a likely theme based on recurring motives. Their model encodes only interval content, ignoring rhythm. Like Chai and Vercoe, they did not attempt to use musical domain knowledge to enhance their algorithm. The algorithm also reports likely theme statements, though it ignores fragments and partial statements, and has trouble with transformations like the "tonal answer."

Method

Our general approach is to create a model analogous to a Profile HMM for each theme based on the notes it comprises. Then we compare note subsequences from the music to our HMMs and a random model to find the best classification.

In musical analysis, events are nearly always measured in a relative way, apparently to match human perception of music. If a piece of music consists of exactly two consecutive notes, the first one will have a neutral interpretation, and the second will be analyzed in relation to the first, in terms of pitch, length, volume, timbre, etc. When we take a MIDI file as input, we need

a simple encoding that captures the most relevant information about each note in relation to the previous one. Since we are concerned with linear (horizontal) analysis, we will mostly ignore vertical harmonic considerations and analyze each voice separately. We define a note n as the ordered pair (h, r), where

- if n is the first note in the voice, $n = (0, 1)$;
- otherwise, h is the difference ("interval," in music theory) in half-steps between n and the previous note, and r is the ratio between the length of n and the previous note.

In our implementation, the MidiReader class takes a MIDI file and returns a Score object with a list of voices, where each voice is a list of Note objects. Class Note records h and r as well as absolute MIDI data for use in the GUI. There is one more value that we use in constructing HMMs, though we do not need it for analysis once we have the models. Let the musical scale function value $f(n)$ be the half-step difference modulo 12 between the absolute pitch of n and the tonic pitch (given as a parameter; by default, the tonic is inferred as the lowest final pitch). There are 12 half-steps in an octave, and it is a basic principle of music theory that notes separated by one or more octaves have identical scale functions.

Theoretically, there are infinitely many possible notes as we defined them. However, a single piece of music has a finite number of notes, and some of them are duplicated many times. Once we have a score, we create an Alphabet object that maps notes to integer symbols. The Alphabet also allows a special "null" symbol to represent notes that do not occur in the score and an "end" symbol used by the HMMs to terminate a theme. The "random model" emits symbols from the alphabet in proportion to their frequency in the score.

The Theme class holds a sequence of notes that will be used to generate an HMM in the

ThemeProfileModel class. The HMM algorithms are handled by Jahmm v. 0.6.1, a free "Java HMM" library. A typical Profile HMM has a column of states for each match position, including match, insert, and delete states. The states in a single column transition only to states in the next column (or to the end state), keeping the number of states linear in the number of match positions. We adopt a similar model, but insert and delete states are not as appropriate for our analysis, so our columns will look different. Our model will be based on a single sequence, so there is no multiple alignment. Instead, we use *a priori* domain knowledge to predict musically likely substitutions for each sequence position.

As with a Profile HMM, we begin with a linear chain of states, creating one state per symbol in the theme sequence. We ignore the first symbol, as it will always depend on the note before any given theme statement. For each position, we define two kinds of substitutions: first- and second-order.

A first-order substitution replaces a single note tuple with another one. This is easily implemented by adjusting a state's emission distribution to include symbols other than the match symbol. We include the following emission types in our distributions, along with the default distribution parameters:

- Match (60%)—the most likely emission.
- Tonal answer (10%)—the musical motivation of the tonal answer is beyond the scope of this paper. We use this definition:
  - When $f(n_{i-1}) = 0$ (the previous note was the "tonic"), a tonal answer will probably shift n by –2 halfsteps;
  - When $f(n_{i-1}) = 7$ (the previous note was the "dominant"), a tonal answer will

probably shift n by +2 halfsteps;

◦ Otherwise, there is no tonal answer substitution, so emit the null symbol.

• Same rhythm (15%)—symbols for notes with same r but different h.

• Same pitch (5%)—symbols for notes with same h but different r.

• Other (10%)—all symbols not described above.

When more than one symbol in the alphabet fits one of these categories, we distribute the symbols uniformly within each category. The "other" category acts like a pseudocount, allowing for occasional unexpected emissions and noise.

Sometimes, a theme statement may replace individual pitches; for example, if the mode changes from major to minor, each pitch with a scale function of 4 half-steps above tonic (the "mediant") will be replaced by a pitch one half-step lower. Since our note model encodes intervals, a pitch substitution will affect two h values: the note before the pitch substitution, and the note after. If we allow for pitch substitutions, then each event depends not only on the previous one, but also the event before that. The score then represents a second-order Markov process, and we say that pitch substitutions are second-order substitutions. While the first-order substitutions could be represented with an emission distribution, the second-order substitutions require additional states.

To generate the additional states, we first determine the number of pitch substitutions for each state, which depends on the function of the theme match note at each position. For each match note n_i, compute $f(n_i)$. Then look up likely substitutions in a table P given as a parameter. Let $P(f)$ be the set of likely substitutions for scale function f. Each substitution is a pitch shift Δh in half-steps.

Table 1: Default pitch substitutions by half-step scale function.

f	0	1	2	3	4	5	6	7	8	9	10	11
$P(f)$	{0}	{0, 1}	{0, -1}	{0, 1}	{0, -1}	{0, 1}	{0, -1}	{0}	{0, 1}	{0, -1}	{0, 1}	{0, -1}

For all f, $0 \in P(f)$ so that we include the match state (no substitution). At note n_i in the sequence, $P(f(n_i))$ gives the likely pitch substitutions. The default table uses at most 2 substitutions, but there could be more if desired (for example, we could allow $P(6) = \{0, -1, +1\})$. Then we need $|P(f(n_i))|$ states for the ith position, if the previous position had no pitch substitutions (or $i = 1$). However, if note n_{i-1} did use pitch substitution states, we need one set of substitution states for each substitution state at position $i-1$: each state must "undo" the previous pitch shift and then add its own shift. Thus the number of states we need in the ith position is

$$|P(f(n_{i-1}))| \cdot |P(f(n_i))|,$$

which is never greater than 4 when using the default pitch substitution table. For the jth substitution in position $i-1$ and the kth substitution in position i, pitch shift is

$$\Delta h_{i,j,k} = -P_j(f(n_{i-1})) + P_k(f(n_i)).$$

We allow a simple distribution to govern transitions from one position to the next. The distribution is uniform among all pitch substitution states, except for the zero substitution (match state), which receives extra weight (3 times as much, by default).

Pitch substitutions are handled by the StateColumn class in our implementation. Each StateColumn is a group of states that correspond to a single position in the theme. The ThemeProfileModel class first creates a sequence of StateColumn objects for a given theme to handle second-order substitutions, then the first-order substitutions are applied to the emission distribution of each state. In order to model theme fragments as well as statements, we let the

model begin in any state, though states in the first column are given significantly more weight (default 25% of the distribution) and some of the transition distribution in each state is redirected to the end state (default 10%).

Once we have one or more theme models, we can "annotate" the score by labeling notes that could be generated by a model with high probability. The analyze() method returns a list of MotiveBracket objects that represent score annotations. Since we assume that the voices are independent lines, we can analyze each voice separately. The heart of the analysis is done in bracketVoice(). This method scans the voice repeatedly, looking first for whole theme statements, then for shorter fragments, until the minimum subsequence length (default 3) is reached. A "bracket" annotation is made if the log-odds score for subsequence is greater than the score from the random model by a given threshold (default 0.15). Once a statement is bracketed, the bracket will not be erased. However, when marking fragments, the algorithm will replace a long bracket with one or more shorter brackets if at least one of the shorter brackets has a better log-odds score than the longer one. All of the default parameters were chosen manually based on expert knowledge and experimentation; they can be adjusted at runtime.

Our model captures many common contrapuntal techniques, including transposition, tonal answer, modal alteration, augmentation, diminution, fragmentation, and stretto. Inversion, the transformation of a theme by inverting each interval, is typically applied to an entire theme or fragment (not to individual notes). To model inversion, the analyze() method creates a second "theme" by taking each of the original themes and replacing each note $n = (h, r)$ with $m = (-h, r)$. The algorithm will then automatically choose whether a sequence is better described by a theme model, its inversion model, or neither. Retrograde, reversing the order of pitches in a theme,

could be modeled with a similar method, but this technique is far less common and often difficult to perceive with the ear, so we chose not to generate "retrograde" models.

The algorithm described above is our "standard" analysis. We recommend that it be given a theme model for the subject and any countersubjects in a work, excluding the "resolution" note of each (since the duration of this note will often be different in each statement).

We also implemented an optional "motive search" meta-analysis algorithm. When writing a fugue, the composer may take certain fragments of the subject as "motives" and reuse them more frequently and in more creative ways than other fragments. The motive search first performs a standard analysis, then examines the list of MotiveBracket objects to see if certain fragments appear more often than some threshold (default 3 times). Such a fragment is labeled a "motive" and given its own theme model. Once a set of motives has been determined, the analysis is performed again, with the original themes and the motive models. This second run may find additional manipulations of a motive that were initially deemed unlikely because of the log-odds "penalty" for beginning at a later position in the theme or ending early.

To increase the utility of our analysis software, we implemented a simple GUI that can display MIDI files and analyses in a piano-roll-like format similar to the diagrams of Lubiw and Tanur, where each note is a long rectangle (Figure 1). The GUI has access to model parameters, and the user can define themes by clicking on notes in the score (Figure 2). A simple playback feature is provided so the user can verify the analysis aurally. The user can scale the display of the MIDI file and add barlines for easy reference. The ouput (Figures 3 and 4) can be saved as a PNG image or described in a text window for convenience.

Evaluation is not straightforward, since the kind of analysis our algorithm performs is somewhat subjective. Since the algorithm incorporates our own music-theoretic domain knowledge and biases, we concluded that evaluation should be performed by an independent expert. We approached Dr. Stanley Kleppinger, Assistant Professor of Music Theory at the University of Nebraska—Lincoln, and he agreed to compare program output against his own analysis of several fugues. Dr. Kleppinger and this author had no relationship prior to this evaluation (beyond working at the same school). We provided him the output from the following eight fugues for evaluation:

- Bach, J.S. *Das Wohltemperirte Clavier*, Book 1, Fugue Nos. 1, 2, 4, 7, 9, 16, and 20.
- Shostakovich, Dmitri. *Twenty-four Preludes and Fugues*, Op. 87, Fugue No. 1.

For each fugue, we specified a subject and countersubject as the "themes" and used the default model parameters, with two minor exceptions. The fourth Bach fugue has an exceptionally short subject (only four notes, excluding the resolution), so we disabled the fragmentation analysis, allowing only complete statements or transformations of the subject. The seventh fugue has a trill in the middle of the subject, realized as twelve distinct MIDI events; we chose to specify the subject as two themes separated by the trill, labeled "Subject" and "ArpeggioMotive" in the analysis. This avoids skewing the model by including repetitive states for a series of tiny "notes" that are aurally not very relevant.

Results

Dr. Kleppinger offers the following evaluation of the software:

"The software demonstrates an aptitude for recognizing fugal subjects and countersubjects at a level one might expect of a student who has become at least somewhat

familiar with this repertoire. It has no trouble identifying verbatim or slightly-altered versions of these themes (including inversions), and is able to make sound judgments about the relatedness of other contrapuntal material to these themes. Occasionally it puts forth observations that might have escaped my attention! More rarely, it posits as similar thematic materials that I simply can't correlate. In summary, though, this software would perform quite well in an undergraduate-level course exploring fugal forms for the first time, as a great part of students' work at this level is identifying statements of the subject and countersubject—a task this software performs quite well."

Dr. Kleppinger's comments on the program's analysis of each fugue are included in the appendix.

Conclusions and Future Work

Using hidden Markov models, we created an effective tool for analyzing fugues. Given a theme, our algorithm uses prior musical assumptions to model likely transformations and find approximate matches in a piece of music. The analysis performed is (although not perfrect) comparable to the work of competent counterpoint students, and the results were validated by an independent expert. Evaluation was concentrated on Bach fugues, but due to the musical generality of the model, the software could conceivably be applied to any music in the Common Practice tradition where one or more voices can be clearly defined. Future research could further explore the application of the software. Model parameters could be tuned, either by an expert or through another machine learning algorithm. For some longer subjects with many short notes, a hierarchical HMM might be appropriate to represent musical gestures at a more abstract level. Such a model might also be more computationally efficient. However, our current

implementation should provide a reasonable starting point for the analysis of many fugues and other pieces.

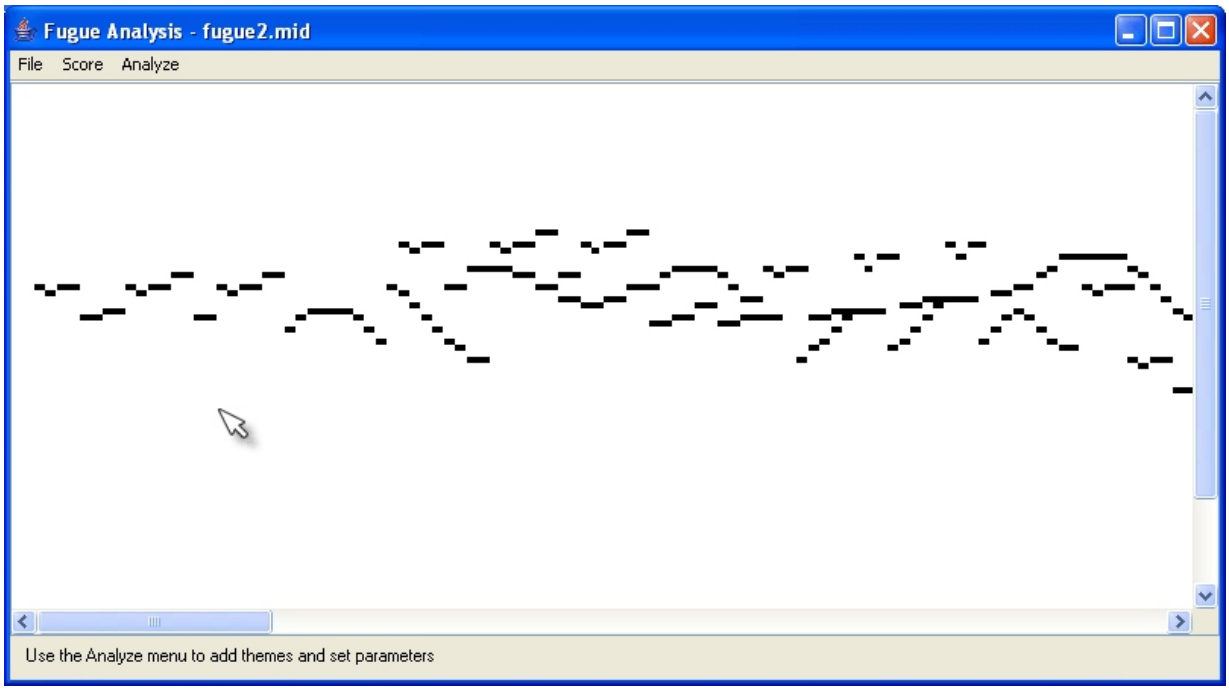

Figure 1: Fugue No. 2 from J.S. Bach's *Well-Tempered Clavier*, Book 1.

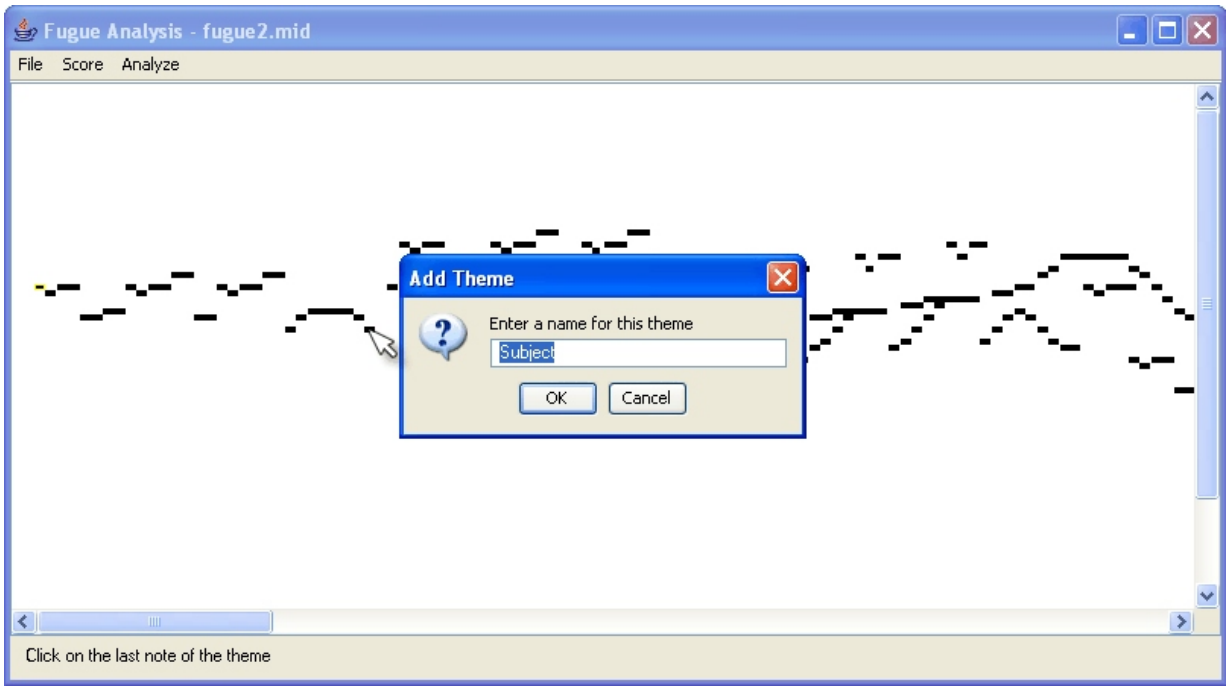

Figure 2: Defining the subject.

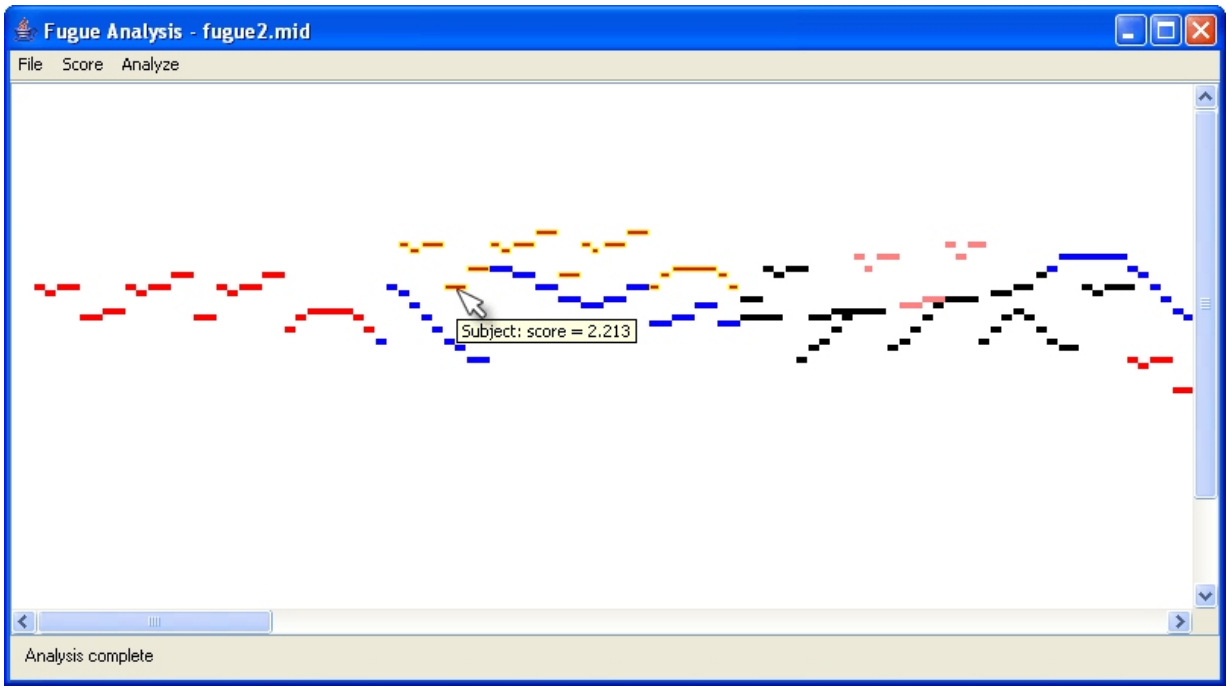

Figure 3: Analysis complete.
The tonal answer is highlighted as the mouse points at it.

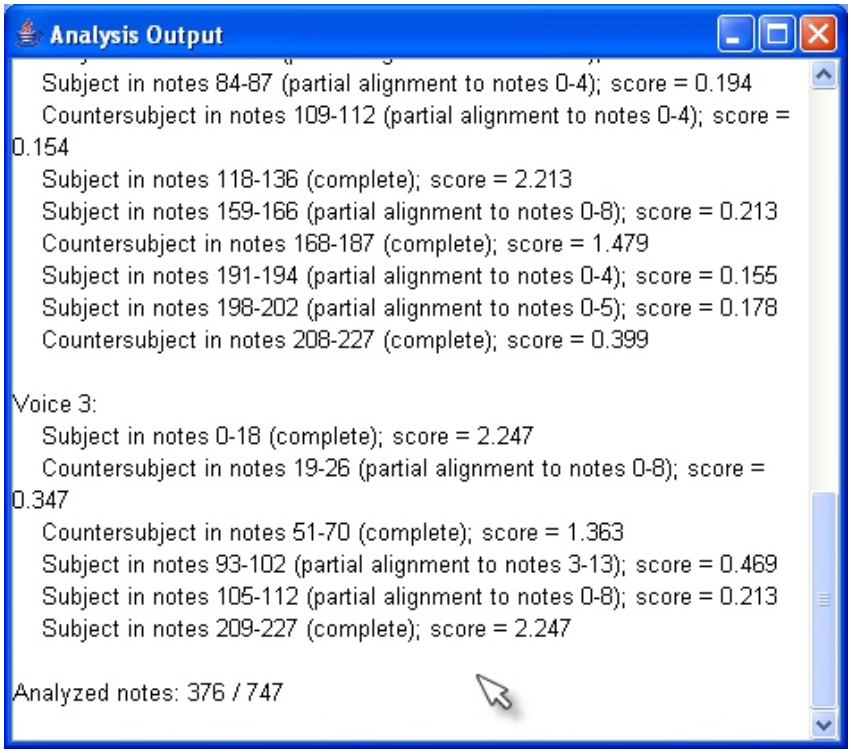

Figure 4: Text output.

References

Chai, Wei and Barry Vercoe. 2001. Folk music classification using hidden Markov models. In *Proceedings of International Conference on Artificial Intelligence.*

Lubiw, A. and L. Tanur. 2004. Pattern matching in polyphonic music as a weighted geometric translation problem. In *Proceedings of the 5th International Symposium on Music Information Retrieval.*

Radicioni, D. P. and M. Botta. 2006. A methodological contribution to music sequences analysis. In F. Esposito and Z. W. Ras (Eds.), *Foundations of Intelligent Systems*, in *Proceedings of the 16th International Symposium on Methodologies for Intelligent Systems.*

Appendix: Expert Commentary

This author has added annotations in brackets to explain some of the music theory in the commentary.

Bach Fugue No. 1

Your user-input suggests a CS [countersubject]—I don't think this fugue has one. (This is borne out by the relative absence of CS statements in the analysis.)

Picks up false lead into m. 16 (Voice 1) AS false. Nice.

[While all fugues have a "subject," the counterpoint against the subject in the second entry is typically analyzed as a "countersubject" if an only if it is used extensively in the rest of the fugue. Since this information is not known in advance, we used the naive assumption that there is always a countersubject. Dr. Kleppinger observes that the algorithm was correct to largely ignore this "theme" input.]

Bach Fugue No. 2

Why not inferred theme at Voice 1, m. 5 (fragment Eb-D-Eb)? Subsequent sequential statements were found.

I think I'd start CS AFTER downbeat of m. 3 (and end S [subject] on that downbeat), but this is a matter for user consideration rather than an issue with the algorithms.

Why not inferred theme at Voice 3, beginning of m. 10? (Recognized earlier in the same sequence—Voice 3, m. 9, first half.) Is it because of the missing first upward leap? (Which, as I imply above, I don't think is really part of the CS.)

Interesting that last pedal-tone statement is "inferred."

Bach Fugue No. 4

Odd inferred S in mm. 26-27.

Misses S in m. 38, voice 2. This S has a cadential leap in m. 41; does this confuse the algorithm?

Misses S in m. 44, voice 2.

Seems confused by S in voice 2 beginning in m. 54. Cadential leap again?

Short subject causes frequent inferred entries that don't match intuition. I would think this would be difficult to "teach!"

[This fugue shows how the program has more trouble when the subject is extremely short.]

Bach Fugue No. 7

Voice 1, mm. 10-12: Why start S so early? What leads it to begin S on the F in m. 10 rather than on the Eb on the downbeat of m. 11?

Notices augmented/sequenced version in Voice 3, m. 16. Good!

Also nicely catches little diminution in m. 36.

Bach Fugue No. 9

Picks up on emphasized inversions of subject's opening whole step—e.g., m. 5, middle voice; mm. 13-16, middle voice. But I'm not sure what it's latching onto in mm. 6-7 in relation to the subject! I don't see the connections to the subject the output suggests in mm. 6-7.

Bach Fugue No. 16

Measure 7, top voice—mock stretto is not recorded.

Measure 10, bottom voice—false lead is not recorded.

Throughout, program seems to prefer identifying the tail motive in isolation, but NOT the

head motive. [This could probably be corrected by adjusting the model parameters.]

Bach Fugue No. 20

m. 10 (really m. 11), tenor voice—little variation in middle of subject causes program to see “inferred” [partial] entry rather than “true” [complete] entry. (I wonder what it'd do with the last set of subject entrances in the F-major fugue, which all exhibit the same new diminution.)

Inversion in m. 17 recognized as "inferred" entry—but not m. 21 lowest voice (also inverted).

Does well overall in recognizing inversions.

Shostakovich Fugue No. 1

Most interesting find by the program here is the final entry in the top voice (m. 98; numbered in output file as m. 99). This entrance is augmented AND begins without the telltale ascending fifth that a listener might focus upon.

On the other hand, the output file doesn't take note of the counterpoint that uses fragments from the subject in sequence—e.g., the extension in the bottom voice in mm. 55-57.