Automatic Acquisition of Counterpoint from *cantus firmus*
(with Conditional Random Field & Belief Propagation)

Seok Hyun (Eric) Hong[1]
Adam Krebs

for Machine Learning

* * *

ABSTRACT

Counterpoint in music refers to a musical like that sound very different and move independently from the main melody called *cantus firmus* but sound harmonious when played simultaneously. Blossomed in the Baroque period, counterpoint is most prominent in Bach's work such as English Suite. Austrian music theorist Johann Fux later systematized the rules of counterpoint into 5 sets (called *species*)[2].

The highly systemic and technical aspects of counterpoint allow an investigation on an unsupervised machine learning system that generates a set of the most probable counterpoint note of its respective *cantus firmus* note mapped one-to-one. One effective approach to achieve this goal is a factor graph based learning system with belief propagation for maximum marginal probabilities[3].

We concluded that merely counting the number of rules the machine correctly identified and followed was meaningless in this context because the system will always produce a note from the domain confined by a set of rules. Furthermore, it is impossible to quantitatively assess it because there are more than one possible admissible outcome. The results were rather qualitatively analyzed by the project's two principal investigators (Hong and Krebs) and any disputes were settled by a domain expert[4]. In the end, we provide the accuracies of three trials on the test set with randomized orders[5].

* * *

[1] Requirement fulfillment for a grade of Incomplete from Fall 2010
[2] Explained more in detail in Section 1 (Introduction) and Section 4 (Approach)
[3] Explained more in detail in Section 5 (Algorithms) and Section 6 (Experimental Setup)
[4] Sluyter, Eric. Music Technology, Steinhardt School at New York University
[5] Explained more in detail in Section 7 (Evaluation)

0. Table of Contents

1. Introduction

Counterpoint in music refers to a musical like that sound very different and move independently from the main melody called *cantus firmus* but sound harmonious when played simultaneously. Blossomed in the Baroque period, counterpoint is most prominent in Bach's work such as English Suite. Austrian music theorist Johann Fux later systematized the rules of counterpoint into 5 sets (called *species*):

1. Each note in every added part (parts being also referred to as *lines* or *voices*) sounds against one note in the *cantus firmus*.
2. Two notes in each of the added parts work against each longer note in the given part.
3. Notes move against each longer note in the given part.
4. Some notes are sustained or suspended in an added part while notes move against them in the given part, often creating a dissonance on the beat, followed by the suspended note then changing to create a subsequent consonance with the note in the given part as it continues to sound.
5. The other four species of counterpoint are combined within the added parts.

In this investigation, we focus on the First Species (called *Note against Note*) for its highly technical and mechanical nature. Fux defines it as:

1. Begin and end on either the unison, octave, or fifth, unless the added part is underneath, in which case begin and end only on unison or octave.
2. Use no unisons except at the beginning or end.
3. Avoid parallel fifths or octaves between any two parts.
4. Avoid moving in parallel fourths.
5. Avoid moving in parallel thirds or sixths for very long.
6. Attempt to keep any two adjacent parts within a tenth of each other
7. Avoid having any two parts move in the same direction by skip.
8. Attempt to have as much contrary motion as possible.
9. Avoid dissonant intervals between any two parts.

We transform these written rules into a set of logical statements feasible for our coding environment. This set of rules is implemented in `rules.py`, which will be discussed more in depth in Section 4 (Approach).

The goal of this research is to develop an unsupervised learning system that can (1) learn the first species rules of counterpoint from given input data set of *cantus firmus* and counterpoint pair, (2) fill in a missing counterpoint given its *cantus firmus*, and (3) even generate the best sequence of counterpoint line from its *cantus firmus* line.

2. Research Environment

2.1. System

Host environment is Ubuntu 11.04 with 3.1 GHz i3 Quad Core with 4 GB of RAM

2.2. Language

Main language of choice is Python for its rapid prototyping ability and abundant libraries for music processing. We supplemented it with Bash script for I/O file processing.

2.3. Libraries + Dependencies

- Numpy[6]
  Scientific computing for Python including a powerful N-dimensional array object, broadcasting functions, tools for integrating C/C++ and Fortran code, useful linear algebra, Fourier transform, and random number capabilities.
- Scipy[7]
  A collection of high level science and engineering modules including statistics optimization, numerical integration, linear algebra, Fourier transforms, signal processing, image processing, special functions and more.
- mingus[8]
  A package for Python used by programmers, musicians, composers and researchers to make and investigate music including intervals, chords, scales and progressions.
- Monte[9]
  A Python framework for building gradient based learning machines including conditional random fields.
- Anthony Theocharis's Counterpoint Library[10]
  This system aims to provide a simple mechanism for the input and evaluation of music according to theoretical rules especially counterpoint. Theocharis modified Mingus library (https://raw.github.com/anthonyt/counterpoint/master/anthonys-mingus-patch.diff) to suit the needs of his project.

  The code basis of this research is an extension of Theocharis's library. The counterpoint library does not employ any machine learning algorithms; it rather evaluates if the counterpoint notes are valid in a given input midi file using a rule-based approach. We have significantly modified and augmented the existing library suitable for our research environment and implemented several machine learning approaches to generate counterpoint notes.

[6] http://numpy.scipy.org
[7] http://www.scipy.org
[8] http://code.google.com/p/mingus/
[9] http://montepython.sourceforge.net/
[10] https://github.com/anthonyt/counterpoint

These implementations are discussed more in detail in Section 4 (Approaches) and Section 6 (Experimental Setup).

3. Data Set

One of the biggest challenges for this research was getting optimal yet large enough data set (to be statistically valid) for training and test as each input has to satisfy very specific set of rules:

- Each input must have a set of counterpoint notes
- Every counterpoint note must belong to *first species*
- For every *cantus firmus* note, there must be only one counterpoint note (one-to-one)
- (Preferred) Each input only consists of whole notes

Most songs adopt combinations of different species to enrich the harmonies and the progressions. As a result, we started with a large number of candidates and sieved them down to a smaller optimal input data set.

3.1. Input

Bach's English Suites are the epitomes of counterpoint. We obtain a complete collection of MIDI files (sequenced by Gary Bricault[11]) of Bach's English Suites No. 1 through No. 6 (*BWV 806-811*) on Dave's J. S. Bach Page[12].

[Filtration]
list = { }
foreach (MIDI file m in MIDI list):
    Using mingus, splice m into subsets of length 4 bars *(12.4 notes on average)*
    foreach(Subset s in subset list):
        list.append(s) if {
        s contains counterpoint
        counterpoint in s is first species
        counterpoint in s is one-to-one mapping }

From this process, we obtained 236 sets

[Preprocessing]
Using mingus:
    Convert MIDI file into mingus object
    Separate *cantus firmus* line and counterpoint line

[11] http://www.garybricault.com/
[12] http://www.jsbach.net/midi/midi_english_suites.html

Assemble a vector list of (*cantus firmus*, counterpoint)

[Training Data Set]
Shuffle the list of inputs
Assemble a vector list of (*cantus firmus*, counterpoint) from the first 70% of the list

[Test Data Set]
Assemble a vector list of (*cantus firmus*, counterpoint) from the last 30% of the list
foreach (Subset s in list):
    [Test Task 1[13]] Take counterpoint and drop a note
    [Test Task 2[14]] remove counterpoint
Divide the new list into 3 test sets

Out of the 236 eligible sets:
    Training: 166 sets (70.3 %)
    Test: 70 sets (29.7 %) divided into 3 test sets (23/23/24 sets, randomly chosen)

3.2. Output

Output per each test input consists of both a MIDI file and a plain text file of:
[Test Task 1] Completed counterpoint line with the gap filled in
[Test Task 2] Best counterpoint trajectory

MIDI files were also generated using mingus for qualitative analysis.

4. Approach

We present high level description of our approach here. Please refer to Section 5 (Algorithms) for the details on theoretical/algorithmic aspect and Section 6 (Experimental Setup) for more information on our implementation.

4.1. Overview

A Conditional Random Field (CRF) is an energy-based factor graph in which the factors are linear in the parameters shallow factors and the factors take neighboring output variables as inputs. CRF is a type of discriminative undirected probabilistic graphical model used to encode known relationships between observations and construct consistent interpretation. Therefore, it is often used for labeling/parsing of sequential data.

[13] Filling-in-the-gap Task
[14] Finding-the-best-counterpoint-trajectory Task

For example, one can construct a CRF that takes a sentence X (x_i word) and generates a sequence of POS Y (y_i POS for x_i). In our research, we adopt this graphical model and represent our problem as a *cantus firmus* line X (x_i each *cantus firmus* note) and a counterpoint line Y (y_i counterpoint note for x_i).

Each counterpoint rule in first species can be considered as a factor.

Inference using sum-product algorithm results in a most probable counterpoint note. On the other hand, we can obtain the most probable sequence of counterpoint note by employing max-product (analogous to Viterbi algorithm for hidden Markov models).

4.2. Note Representation

`Note` class contains necessary information as fields. Some important fields are:

- isFirstOrLast — specifies if a given note is either first or last
- isRest — specifies if a given note is a rest
- pitch — representation of pitch

4.3. Rule Representation

As rules are factors in our CRF model, these are represented as functions that take a list of notes as a parameter, determine if a list satisfies the rule and returns an evaluation.

The following is an example of a rule function (a factor):

```
def illegal_strong_beat_horizontal_intervals(a_list):
       allowed_intervals = ['1', 'b2', '2', 'b3', '3', '4', '5', 'b6', '6']
       intervals = strong_beat_horizontal_intervals(a_list)
       return [x for x in intervals if x[0][0] not in allowed_intervals]
```

4.4. Training

During training, we feed the model with a list of *cantus firmus* notes, its corresponding list of counterpoint notes and factors (counterpoint rules from first species). The model examines each (*cantus firmus*, counterpoint) note pair, considers its properties (position, pitch, etc.) and determines (i.e. returns) a weighted list of possible rules applied to the pair, adding each weight to its respective factor. In the end, the model learns the conditional distribution between a counterpoint note and rules from the training data.

4.5. Test

[Test Task 1 - Filling-in-the-gap]

- Inference: sum-product belief propagation algorithm (analogous to forward-backward algorithm in hidden Markov models) to compute the posterior marginal of a missing hidden variable using dynamic programming
- Decoding: determination of the most likely note (random selection from a range)

[Test Task 2 - Best-trajectory]

- Max-product belief propagation algorithm (analogous to Viterbi algorithm in hidden

Markov models) to find the most likely sequence of counterpoint notes

5. Algorithms

5.1. Conditional Random Field (CRF)

[Definition]
Graph $G = (V, E)$ such that $Y = (Y_v)_{v \in V}$ so that Y is indexed by the vertices of G
(X, Y) is a CRF when conditioned on X, random variable Y_v obeys Markov property $P(Y_v \mid X, Y_w, w \neq v) = P(Y_v \mid X, Y_w, w \sim v)$ where $w \sim v$ means w and v are neighbors in G[15]

[Inference]
Inference is usually intractable but if a CRF is a linear chain, then an effective message propagation yields exact solutions. In our case, counterpoint note prediction task is analogous to sequence modeling - a chain graph. Therefore, using a belief propagation algorithm like sum-product algorithm and/or max-product algorithm coupled with dynamic programming produces good results.

[Parameter Learning]
Learning parameter θ is done by maximum likelihood learning for $p(Y_i \mid X_i\,;\,\theta)$

Joint Density Function	$f(x_1,\ ...,\ x_n \mid \theta) = f(x_1 \mid \theta) \cdot f(x_n \mid \theta)$
Likelihood	$f(x_1,\ ...,\ x_n \mid \theta) = \prod_{i=1}^{n} f(x_i \mid \theta)$
Log-likelihood	$\sum_{i=1}^{n} \ln f(x_i \mid \theta)$
Maximum Likelihood Estimator	$\theta_{mle} = \arg\max_{\theta \in \Theta}(\theta \mid x_1,\ ...,\ x_n)$

[Factors (Feature Functions)]
Conditional dependency is feature function measurements on the input sequence that partially determine the likelihood of each possible value of Y_i

5.2. Sum-product Algorithm

Estimation of marginals

[Message]
Messages are similar to likelihoods.
A high value of message $m_{ij}(x_j)$ means that node i “believes” the marginal value $P(x_j)$ to be high[16]

[15] Lafferty, McCallum and Pereira (2001)
[16] Coughlan, James (2009) “A Tutorial Introduction to Belief Propagation”

[Message Update]

$$m_{ij}^{new}(x_j) = \sum_{x_i} f_{ij}(x_i, x_j) g_i(x_i) \prod_{k \in Nbd(i) \setminus j} m_{ki}^{old}(x_i)$$

[Belief]

$$b_i(x_i) = g_i(x_i) \prod_{k \in Nbd(i)} m_{ki}(x_i)$$

5.3. Max-product Algorithm

Estimation of the state configuration with maximum probability

Max-product algorithm is similar to sum-product algorithm:
[Message Update]

$$m_{ij}^{new}(x_j) = \max_{x_i} f_{ij}(x_i, x_j) g_i(x_i) \prod_{k \in Nbd(i) \setminus j} m_{ki}^{old}(x_i)$$

(cf) sum is replaced by max

[Belief]

$$b_i(x_i) = g_i(x_i) \prod_{k \in Nbd(i)} m_{ki}(x_i)$$

(cf) this is now a scoring function whose maximum points to most likely state

6. Experimental Setup

6.1. Data Set

Prepare training and test data sets as described above in Section 3

6.2. Framework

Step 1: Take input X_i and construct conditional random field

[Training]

Step 2: Compare x_i and y_i $(\in X_i)$ and analyze considering different properties
Step 3: Determine possible rules (factors) and weigh them
Step 4: Add weights to each factor
Step 5: Repeat Steps 2-4 for x_{i+1} and y_{i+1}
Step 6: Repeat Steps 1-5 for X_{i+1}

[Test]

[Task 1]

Step 2: Analyze $[\,(x_1,\,y_1),\,(x_2,\,y_2),\,...,\,(x_{N-1},\,y_{N-1})\,]$

Step 3: Calculate sum-product for this propagation

Step 4: Find the maximum likelihood function for x_N

Step 5: Decode y_N note by choosing a random note from possible notes

(This yields the best counterpoint note)

Step 6: Output to MIDI and text files

Step 7: Repeat Steps 1-6 for X_{i+1}

[Task 2]

Step 2: Take x_i where $i = 1,\,...,\,N$

Step 3: Calculate max-product for this propagation of $x_{i-1} \rightarrow x_i$

Step 4: Decode y_i note by choosing a random note from possible notes

Step 5: Repeat Steps 2-4 for x_{i+1}

(This yields the best counterpoint sequence)

Step 6: Output to MIDI and text files

Step 7: Repeat Steps 1-6 for X_{i+1}

7. Evaluation

The most challenging part of this research was the lack of training and test sets in terms of quality and quantity large enough for a truly meaningful statistical analysis. In addition, the output labels are not always limited to one, but are rather free as long as they meet the specific rules. This makes quantitative analysis very difficult as it lacks clearly defined metrics.

Counting whether or not the output obeyed the rules becomes less relevant in the field of music where harmony is a subjective matter. For the scope of this research, we tried to develop a hybrid evaluation model - an amalgamation of quantitative and qualitative assessments.

7.1. Quantitative Criteria

$$Qt(Y_i) = \left\{ \begin{array}{l} 0, Y_i\, does\ not\ follow\ the\ rules \\ 1, Y_i\, follows\ the\ rules \end{array} \right\}$$

7.2. Qualitative Criteria

$$Ql(Y_i) = \left\{ \begin{array}{l} 0, Y_i\, does\ not\ sound\ good \\ 1, Y_i\, sounds\ good \end{array} \right\}$$

7.3. Method of Evaluation

$$Eval(Y_i) = Qt(Y_i) \cap Ql(Y_i)$$

In words, the evaluation of an outcome is a logical AND of the qualitative and quantitative outcomes. For example, if the qualitative assessment is 1, and the quantitative assessment is 0, then the overall assessment is 0 (incorrect)

7.4. Dispute Resolution

Two primary investigators independently assess each outcome qualitatively. If the two assessments coincide, then this becomes $Ql(Y_i)$. Otherwise, a third domain expert evaluator assesses it and makes it a final assessment.

7.5. Results

We present the results of three test sets performed on two different tasks.

	Task 1				**Task 2**			
	Qt	*Ql*	*Eval*	*%*	*Qt*	*Ql*	*Eval*	*%*
Set 1 (23)	21	20	20	87.0	22	17	16	69.6
Set 2 (23)	18	15	15	65.2	21	19	19	82.6
Set 3 (24)	20	18	18	75.0	18	16	14	58.3
Average				75.7				70.2

Quantitative results are satisfactory for both tasks as the system produces counterpoint notes according to the rules it learned from the training sets

Qualitative analysis shows that the system did much better on Task 1 than Task 2. Fill-in-the-gap task (Task 1) only requires to generate one counterpoint note. Therefore, the generated counterpoint note has much higher chance of harmonizing with the rest. However, Best-trajectory task requires the system to generate a whole sequence, which, to human ear, might sound a lot less harmonious.

8. Conclusion

We used graphical approach to automatically generating counterpoint notes using conditional random fields and two belief propagation algorithms - sum-product algorithm for finding a maximum likelihood marginal probability for a missing counterpoint note and max-product

algorithm for finding a most probable sequence of counterpoint notes.

The results are enough to prove that this approach has a potential and perhaps very efficient. However, the subjective nature of the domain of interest poses a huge barrier for an objective evaluation as well as automation of such processes. For example, what we have been able to accomplish for this research was only the first species of five species rules. The rules in other species are too much subjective and broad to be implemented using a machine.

It is still novel at this stage unless it is supplied with more data and more sophisticated algorithms. However, it is a very promising approach nonetheless.