CS 221 Final Project Progress Report:
The Canon Solver

Bryan McCann (bmccann) & Zachary Yellin-Flaherty (zachyf)

Introduction

Task Definition: Design a system that can solve the **Puzzle Canons** posed by Bach. **Canons** consist of one single-voiced melody that can be transformed in an enumerable set ways to create a poly-textural musical piece that follows all the conventions of classical harmonic practices. It is common to call the proposed melody the **theme,** and the transformations **variations on the theme.**

High Level Solution:

1. Construct a model of musical 'fluency' primarily based on harmonic principles.
2. Frame the search for solutions to Bach's canons as a search problem with costs determined by our Musical Fluency Model.

Musical Fluency: Modeling Musical Harmony in the Style of Bach

Model Construction: To construct our model of musical harmony, we extract features from a large corpus of 150 Bach chorales. We currently extract only bigram, trigram, and tetragram features. This process is analogous to that in natural language processing, but the 'words' in this case are intervals if we have only two voices (two notes at that time slices) and chords if there are more than two voices. A bigram would then consist of a tuple (IntervalA, IntervalB) in the case of two voices or (ChordA, ChordB) in the case of more than two voices.

Because our model is primarily concerned with harmony, we abstract away the notes of the intervals and chords. We only extract the Roman Numeral assignment to the interval or chord determined by the current key. For example, our bigram features look like (BEGIN, I), (I, ii6), (ii6, V), (V, I). this would show that the analyzed music contained the common I-ii - V I progresssion in a major key to begin the piece, with the ii chord in an inversion. The analyzed music might have been CMajor, dminor/F, GMajor, CMajor, but our model ignores those specifics.

Rationale: Modeling only the relationships between chords and intervals allows us to see the higher level trends in the music, observing how Bach creates tension and resolution and plays with harmonic principles.

We do not include unigrams because this too would give too much information irrelevant to the

harmony of the pieces. Bigrams allow the model to capture relationships from one chord to the next, and trigrams and tetragrams capture chord progressions.

Currently, the model considers fluency as primarily grounded in the frequency of these chord and interval N-grams. For example, in the large corpus that we analyzed, Bach used some bigrams upwards of 25 times, whereas most bigrams present only occur once. The model then suggests that these bigrams are preferred over others in musical and harmonic fluency when they are valid options.

Because our ultimate goal is to solve Bach's Puzzle Canons as he would have intended, fluency is entirely derived from Bach's music.

Transforming frequency into fluency: Our model can be viewed in two ways. First, it is a model of harmonic fluency based on frequency, but it is also intended to serve as a cost/reward function. At this point, the function is simple. Consider x to be our musical piece, let phi(x) be our vector of extracted chord/interval features, and let r be our vector of chord/interval features extracted from Bach chorales where the values of each entry correspond to the average frequency of that feature in the selection from which it was drawn. So in the case of x, an entry in phi(x) will correspond to the number of times that N-gram appeared in x. In the case of r, an entry will be the average number of times that feature appeared in the corpus of Bach chorales. Then the cost for that x will be the Euclidean distance from phi(x) to r. This will allow our model to suggest that adhering to Bach's averages is a good indicator that you will be both harmonically and stylistically similar to Bach in solving the Puzzle Canons.

Solving the Puzzle Canons: Searching for the Right Answer

Defining the Canon Problem: We frame the problem of solving a Puzzle Canon as a search problem, on which we can use Uniform Cost Search. The input to our problem is the theme, a single voice melody that we have preprocessed using the Music21 python libraries. Details on preprocessing and our use of Music21 are below.

The start state is a Music21 stream that represents our theme and a Euclidean distance (cost of accepting this state as a solution) of Infinity as the theme itself is not a viable solution.

The possible actions are 12 transpositions for each chromatic interval, n imitations (where n is the number of notes in the original melody so we can have n shifts of the melody), and a mirror, or reversal, of the original melody.

The search will consist of iterating through all of the possible variations of the theme, calculating their cost by using our model of musical fluency, minimizing the cost, and checking whether we are in and end state.

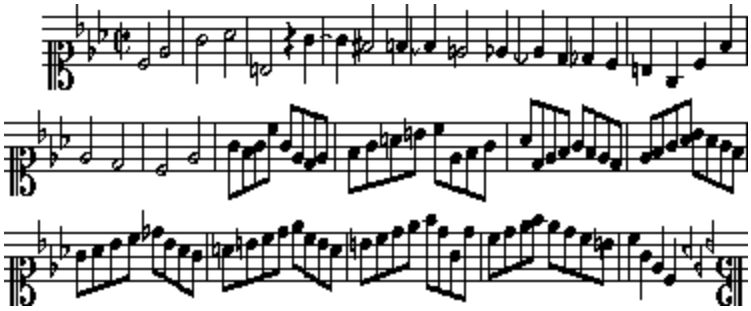
There is one condition for an end state: the move does not make sufficient progress towards Bach's averages according to some tolerance.

Since duration of the entire piece is conserved after each of these transformations, each voice will add harmony to our previous state. As a result, these multiple voices can be reduced to chords or intervals and then analyzed harmonically.

An Example: The Cancrizans

Let's examine an example of inputs and outputs, and how we operate on our data.

The input puzzle we would like to solve could be Bach's "Cancrizans" :



Several hints suggest to musicians and music theorists the solution to this puzzle. "Cancrizans" means "crab" and "walking backward" in Latin. One clef at the end of the piece is backwards. The compositional challenge then is to to play the melody forwards and backwards in harmony. Our algorithm cannot use these hints, and it does not need them.

The first step is to represent the melody in a way our program can understand. For this ability, as well as many other in this assignment, we used an MIT python library called Music21, that abstracts away virtually all of the technical music aspects, and allows us to focus on the model. Virtually the only tedium is inputting the puzzles' themes. We create a part object, and manually add all the notes using code in the form of:

CancrizansTheme.append(note.Note('C4', quarterLength=2))

Music21 has a set of more than 150 Bach chorals, and using reduction functions in the library, we generate a list of harmonic functions and intervals in the form of "...bVIb63 , bVI7b5b2 , bvi54..." and "...7, 9, 9 ,8…" respectively. Note that the chorales typically have 4 parts, which allow us to form chords, and we pick out the soprano and bass parts, (which are almost always the most important parts) to determine the dominant intervals. These lists represent the vocabulary of Bach, and this precisely the vocabulary from which our model extracts features

using the following code (this feature extractor currently leaves out beginning and end tokens, but this is likely to change):

```
from music21 import corpus, roman, key, interval
from collections import Counter

class ScoreFeatureExtractor:
	"""
	Takes in a list of music21.Stream objects;
	returns a feature vector.
	"""
	def __init__(self, streams=None, bundle=None):
		if bundle:
			self.streams = [x.parse() for x in bundle]
		else:
			self.streams = streams
		self.chordProgressions = []
		self.intervalSets = []
		self.chordFeatures = Counter()
		self.intervalFeatures = Counter()

	def parse(self):
		for stream in self.streams:
			reduction = stream.chordify()
			analyzedKey = stream.analyze('key')
			chords = []
			intervals = []
			for c in reduction.flat.getElementsByClass('Chord'):
				c.closedPosition(forceOctave=4, inPlace = True)
				mainInterval = interval.notesToChromatic(c.pitches[0], c.pitches[-1])
				rn = roman.romanNumeralFromChord(c, analyzedKey)
				chords.append((rn.scaleDegree, rn.quality, rn.inversion()))
				intervals.append(mainInterval)
			self.chordProgressions.append(chords)
			self.intervalSets.append(intervals)

	def extractNgrams(self, sizes):
		sizes = [x-1 for x in sizes]
		for chordSet in self.chordProgressions:
			for idx, chord in enumerate(chordSet):
				key = (chord,)
				for n in sizes:
					if (idx < n or (idx + n) == len(chordSet)): continue
					key += (chordSet[idx+n],)
					self.chordFeatures[key] += 1

		for intervalSet in self.intervalSets:
			for idx, interval in enumerate(intervalSet):
				key = (interval,)
				for n in sizes:
					if (idx < n or (idx + n) == len(intervalSet)): continue
					key += (intervalSet[idx+n],)
					self.intervalFeatures[key] += 1
```

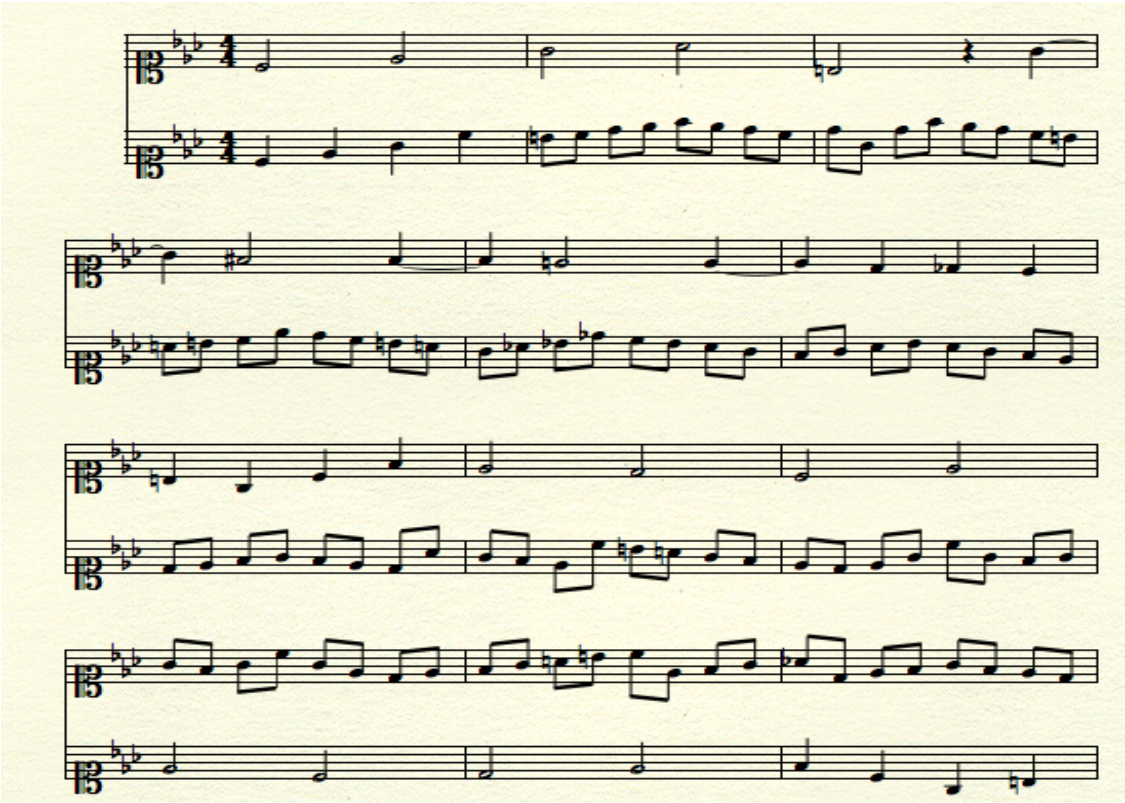
Given these features, we generate our reward function, and we formulate the Cancrizans as one of our test cases, which by default use arbitrary costs and notes that we defined for use in our project proposal:

```
class TestCase(object):
  def __init__(self, name, theme, costs=cu.getSimpleCosts(),
               notes=cu.getSimpleNotes(), verbosity=0):
    self.name = name
    self.costs = costs
    self.notes = notes
    self.theme = theme
    self.verbosity = verbosity
    self.voices = []

  def run(self):
    self.scenario = CanonScenario(self.theme, self.notes)
    problem = CanonProblem(self.scenario, self.costs)
    ucs = UniformCostSearch(self.verbosity)

    print self
    ucs.solve(problem)
    self.getVoices(ucs.actions)
    for v in self.voices:
      print v
...
...
```

where the … represent more code that generates more useful output and extra analysis so that we can determine how changes to our model and algorithm affect results.

Given an input theme, we can then frame our search problem, generate our successor states as previously defined and use the same feature extractor that was included above to extract features from that state. Eventually, we emerge with a solution. For the isGoal() function, we will experiment with different epsilon values forour tolerance. This result represents convergence around a path, i.e. a solution to the puzzle.

Here is the the first 12 bars of the output, which is the correct answer to Bach's puzzle. We easily display the result in Finale Notepad or other MusicXML software using a visualization provided by the music21 library, and the music can also be played back to us with one click.



Unexpected Challenges:

In our previous milestones, we were planning on using the UCI Machine Learning Repository's dataset of Bach chorales. When we further developed our model and our algorithm, we were disappointed to find that this dataset only included one voice; it was essentially useless.

However, this setback was followed up with an incredible discovery. We were pleasantly surprised to discover the MIT Music21 library only this week, which streamlined all of the musical representations. It also included a larger corpus of Bach chorales than the UCI repository had offered, it contained full representations, and it also allowed us to output musical analysis on chords and intervals quite easily, independent of key signature, time signature, etc. Implementing these functions manually would have been very tedious, though not impossible. It was this discovery that allowed us to make so much progress in developing our model and algorithm.

We struggled to define the "fluency" of our cost/reward function for chord or interval progressions as we wanted something analogous to the word seg fluency function for english. Given our large sample of Bach chorales (and other accessible chorales already in this MIT library we can use if we determine Bach is not sufficient), perfecting this function should not prove too difficult.

A large assumption in our definition of fluency and in our model in general is that Bach designed his solutions to the Puzzle Canons to also be incredibly harmonic. We are very confident that this assumption is valid, but if we find that our model does not generate the right solutions to the puzzles, we have an alternative model in mind.

Our backup plan is to train a linear regression model to classify pieces of music as Bach or Not Bach. We discovered Music21's corpus contains thousands of pieces composed by others, and we could train our model to see these as negative examples, and our Bach pieces as positive examples. Then run this on each state to give a score. Possible drawbacks include much more time and the fact that Bach's harmonic style might not differ that much from other's in the corpus. It might also be possible to use K-Means clustering to identify features that are more meaningful if our current feature extraction fails to accurately capture harmonic style.