Composing Fifth Species Counterpoint Music with a Genetic Algorithm

Project for Sound and Music Technology, 2015-2016

Gleb Mineev, Jinfeng Guo and Maguell Sandifort

Game and Media Technology
Information and Computing Sciences

Monday, 27 June, 2016

Overview

The Problem

Counterpoint is:

- the matching between one melody and another melody
- the matching between one melody and multiple other melodies

We only study the basic form, two-part counterpoint with strict rules:

- the matching between two melodies
- input melody is called cantus firmus

The Problem

For counterpoints with strict rules, there are five species

- First species counterpoint: one note against one note
- Second species counterpoint: one note against two notes
- Third species counterpoint: one note against four (or three, six) notes
- Fourth species counterpoint: one note against notes offset
- **Fifth species counterpoint: combination of the first four species**

Challenge

Generating counterpoints is a search problem. For a given melody:

- find the optimal counterpoint melody among all possible melodies
- under the constraints of all rules

The challenge of generating the fifth species counterpoint:

- the high dimension of search space
- the free combination of all previous four species

Related Work

Existing methods:

- Genetic Algorithms
- Probabilistic Markov Chain approach
- Variable Neighbourhood Search algorithm
- Dominance Relation-based method

Research Problem

Our research problems are:

- Can genetic algorithms be used to generate fifth species counterpoint?
- If so, how is the performance of genetic algorithms against other approaches?

The Method

We used a genetic algorithm together with a fitness function

- the genetic algorithm is used by FOOX[1] to generate the first four spieces counterpoints
- the fitness function was proposed by Herremans *et al.*[2], implemented in Optimuse

[1] https://github.com/ntoll/foox

[2] *Composing fifth species counterpoint music with a variable neighborhood search algorithm*

Producing Rhythm Pattern

To form a population, we first calculate an appropriate rhythm pattern

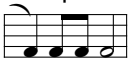
- Simplifies crossover and mutation operators
- Avoid accidental unfeasible rhythm

Output:

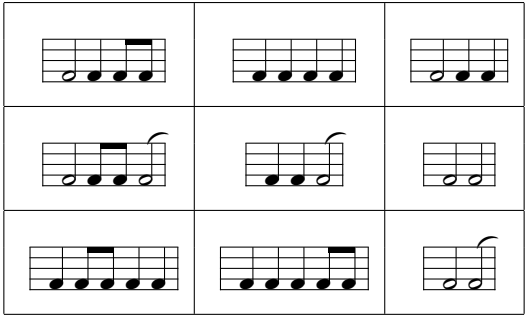
- a rhythmic pattern
- converted into a list of note durations

Rhythm Pattern Rules

Rhythm pattern should satisfy:

- The first measure is 
- The penultimate measure is 
- The last measure is 
- Rhythmic patterns for other measures are selected from the patterns
- No two similar measures in the row are allowed
- When deciding whether two measures are similar or not, we consider two eighth notes as one quarter note.

Rhythm Pattern Rules



List of possible rhythmic patterns

Generating Initial Population

Initial population is created based on previous rhythm patterns.
For each duration, randomly select one that satisfies:

- All notes are higher than the corresponding cantus firmus note.
- For the first note the vertical interval is unison, fifth or octave.
- For the last note the vertical interval is unison or octave.
- Notes on the first beat are vertically consonant.
- Eight notes must move in step.
- The ending is a dissonant suspension.

Mutation and Crossover Operators

Mutation is done by:

- iterating over the chromosome
- changing each single note with a certain probability
- ensuring new note stays within *mutation_range*

We use a standard single-point crossover

- as implemented in FOOX

The Fitness Function

Slightly modified from the original one:

- For horizontal rules use all except for rules 13 and 14
  - Possible contradiction with vertical rule 10
- Combined vertical rule 3 and 4 into a single rule
- Omit several rules already ensured from generation
  - These will always give zero score

Creating Next Generations

The next generation consists of three equal parts:

- the top 1/3 of the previous generation
- non mutated children
- children after mutation

Using Roulette Wheel selection to select two chromosomes for crossover

Creating Next Generations

Normalization is done by:

$$fitness_{norm} = \frac{fitness - fitness_{min}}{fitness_{max} - fitness_{min}} \quad (1)$$

Normalization:

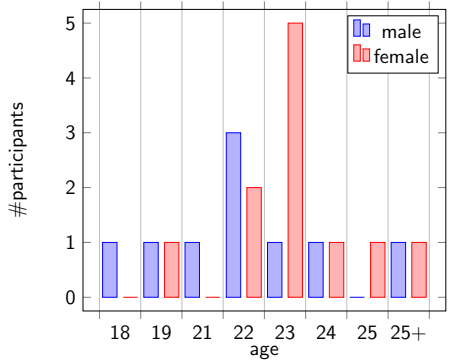
- Allows use of similar selection process for fitness function
- Leads to better results

User Tests

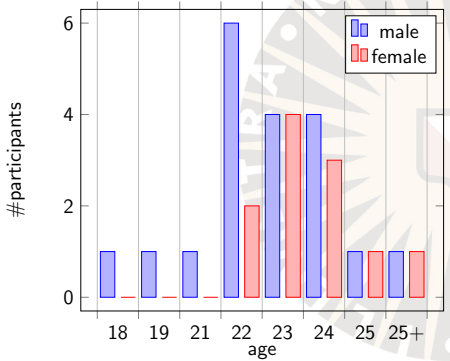
We ran two user tests:

- The first helps determine the optimal parameter set
- The second test compares our result against other results
  - Human composed
  - Random generated
  - Optimuse generated

Participants - Age and Gender

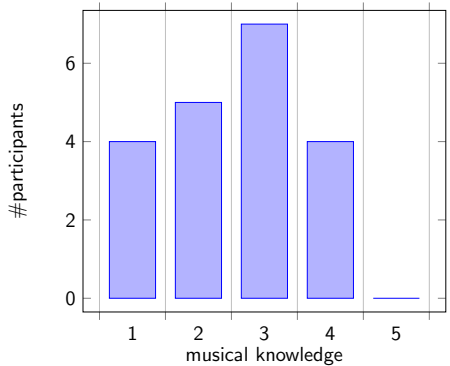


20 participants of test 1

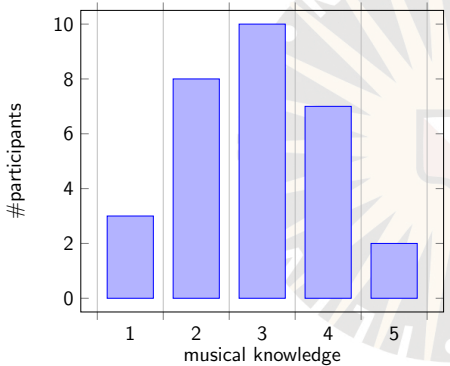


30 participants of test 2

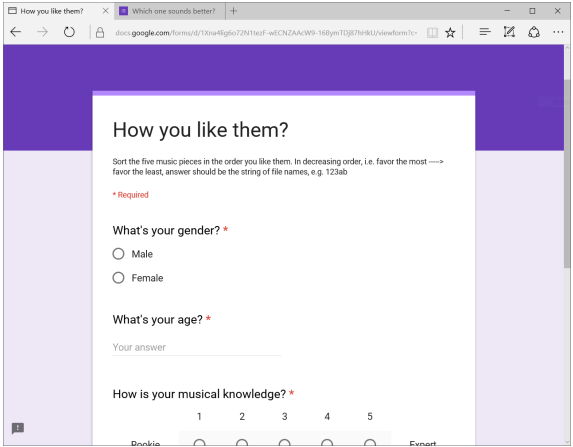
Participants - Musical Knowledge



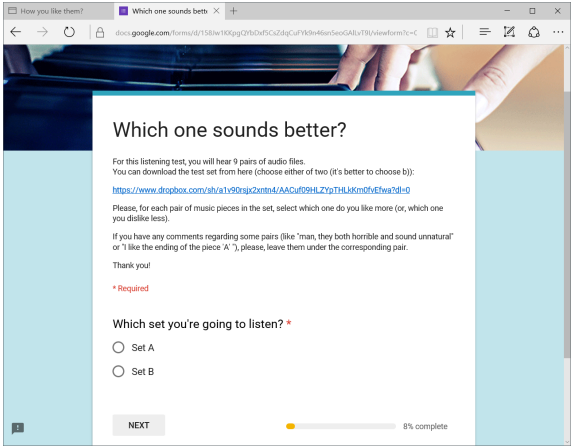
20 participants of test 1



30 participants of test 2



Google Form for the first test



Google Form for the second test

Procedure - Test 1

For the first test:

- Users randomly choose one from three sets
- Each set contains 5 MIDI files
- Users order them from most liked to least liked

Procedure - Test 2

For the second test:

- Users randomly choose one from two sets
- Both sets contain 9 pairs
- Users make their preference by choosing from
  - A is much better
  - A is slightly better
  - Equally good
  - B is slightly better
  - B is much better

Results - test1

Give the ranks a score from 5 to 1

V	2.86	H5	2.76	EQ	3.22
H5	2.48	H18	2.48	H	2.00
EQ	2.10	V	1.71	H5	1.67
H	1.33	EQ	1.52	H18	1.56
H18	1.24	H	1.52	V	1.56

Table: Normalized results after ranking

Results - test1

	H5	H18	EQ	V	H
Test1	2.48	1.24	2.10	2.86	1.33
Test2	2.76	2.48	1.52	1.71	1.52
Test3	1.67	1.56	3.22	1.56	2.00
Total	6.91	5.28	6.84	6.13	4.85

Table: Total scores of test 1

- Scores are fairly spread out
- Use Equal weights

Results Test 2

- Give a score from -2 to 2

Nr.	1	2	3	4	5	6	7	8	9
A	-0.4	-0.2	0.6	1	1	0.6	0	-0.1	-0.3
B	0.1	-0.5	0	0.8	0.8	1.2	-0.1	0.1	0.3

Table: Normalized results after ranking test 2

- Consistently outperform random
- Slightly worse vs human composition
- Equal vs optimuse

Results Test 2

- Compare high and low knowledge results

Nr.	1	2	3	4	5	6	7	8	9
High	-0.4	-0.7	0.3	1.3	1.1	0.7	-0.2	-0.3	-0.8
Low	-0.3	0.5	1	0.3	0.8	0.5	0.3	0.3	0.5

Table: Normalized results of set A high vs low knowledge

Nr.	1	2	3	4	5	6	7	8	9
High	0.5	-0.9	0	0.9	1.2	1.4	-0.2	0	-0.1
Low	-0.8	0.2	0	0.6	0	0.8	0	0.2	-0.6

Table: Normalized results of set B high vs low knowledge

- Low knowledge scores higher on human composition and Optimuse tests
- High knowledge scores higher on random tests

User Feedback

"First piece had a lot less going on than the second."
—some people like ours because of diversity

"B sounds like there are a lot of false notes."
—people actually noticed random samples

User Feedback

"Didn't like either of them, A was less bad than B."
—sometimes results are quite bad

"I like these ones a lot. There actually seems to be some progression in them. Also B is a bit heavier/more emotional."
—more in the direction of a song than just a soundbite, but some are very good

User Feedback

"I liked the part where it went 'Dee dee boodoop da'."

"The overall impression is positive. However, some compositions are 'annoying'"

User Feedback

"The music all sounds like it has been composed locally; short pieces sound okay, but there is no overall structure; no recurring themes. Like how a markov-chain-generated story has valid sentences yet it is not coherent and does not make sense. Also, a decent software synthesizer would make it sound nicer."

Review

- First test had no scores
- Second test had pieces in set order
- No counterpoint experts
- More tests would give more accurate results

Conclusion

- Genetic algorithms can be used to generate fifth species counterpoint
- Our algorithm gains an obvious advantage over random generated counterpoints
- Most human composed counterpoints beat ours
- For shorter pieces, ours is as good as Optimuse
- For longer pieces Optimuse is probably better

Any Questions?

Thank you!