

Generating Fugal Sequences with Clojure, Occam and Reconstructability Analysis

Ryan Spangler

November 28, 2012

Introduction

I have long suspected that Bach's fugues have a high degree of internal relatedness, though I have never had the tools to quantify that until now. This project is a journey to define just how internally related Bach's fugues are, using the means at my disposal (namely Occam and Clojure, along with the theory behind RA). It is an attempt to not just analyze, but also to generate new fugues using the form of old ones. Given the scores of Bach's fugues (48 in all), I tease apart the separate lines into sequences of notes and use these sequences to form predictions about what the next note is, given a series of previous notes. Applying these predictions iteratively to a seed note generates new sequences of notes, which can then be layered and played back. Two supporting Clojure libraries have sprung from this project, *occam* (the clojure version!) and *fuga*. These are open source projects anyone can use to analyze midi files and generate fugues of their own, which I discuss in the next section.

Bach fugues are each unique, but have a similar overall structural philosophy. They each start with a single voice which lays out a theme, and is ultimately joined by a series of voices each performing variations on the original. Each line gets superimposed over the others, and part of the challenge of writing a fugue is tying these lines together in a meaningful and harmonious way (of which Bach is the generally acknowledged Khan of this arena). In order to predict sequences, the common structures and gestures that thematically tie the fugue together have to be extracted and applied on new sequences. Part of the symmetry of the fugal lines is that the theme is repeated, but in a different key or mode. In this way, the information in contained in a relative, rather than an absolute way, and the analysis of the notes data has to reflect this relativity.

Methods

The process of generating new note sequences given a set of existing note sequences takes many steps. First, the note files are parsed and the notes are extracted into a simple key based data format containing a field for the tone, the begin time and the end time. In this form many notes could be happening at once, representing the separate voices of the fugue playing in polyphony. In addition, the onset times are not exactly simultaneous, but fluctuate around a mean with epsilon ± 10 time steps. So all notes are grouped into simultaneous occurrences and these groups are sorted by their mutual onset. Then, separate lines are drawn out of these groups by stepping through each coinciding time step and finding a previous and subsequent note from the previous and subsequent group for each note in that time step. This step is not perfect and note jumps of more than 10 are excluded (arbitrarily, but better to assume too little!) This reduces the series of notes into a handful of chains.

Once the chains are in hand, they can be reduced to a series of snapshots of fixed length relative gestures. This length is variable, so that a variety of lengths can be explored for predictionary power. One of the main uses of Occam in this project was to discover how many previous notes should be taken into account to predict the next note. I explored up to 5, where the defining the data would require two more notes than I wanted to use when doing the prediction. Given a chain of notes [61636663616468] (where 60 is middle C) this can be pivoted on the second to last note (in this case the 64) and all of the previous notes can be masked relative to that one, resulting in $[-3 - 1 + 2 - 1 - 30 + 4]$. The values leading up to the 0 are the previous notes in the chain, the 0 represents the value of the note we will be making the prediction for, and the last value is the dependent variable, the relative shift of the note given what notes came before it. That way 7 notes can be used as a case for 5 previous notes predicting the subsequent note given the current one.

All the chains are reduced to rows in the above relative format by passing a window over them, taking each seven values one after the other. This is written out to Occam format and fed into the Occam web console. In the above case, the independent variables are represented as ABCDE and the dependent variable is Z. We don't need to the pivot note itself because it is always zero by definition: all other notes are considered relative to it for this row. This is the results matrix for five previous predicting notes A-E:

ID	MODEL	Level	H	dDF	dLR	Alpha	Inf	%dH(DV)	dAIC	dBIC	Inc.Alpha	Prog.	%C(Data)	%cover
22	IV:ADZ:BDZ:CZ:EZ	7	9.1287	16996	3019.3988	1.0000	0.91193062	81.1199	-30972.6012	-113206.1360	0.9978	18	85.8521	0.0083
21	IV:AEZ:BDZ:CZ	7	9.1445	14896	2998.9258	1.0000	0.90574728	80.5698	-26793.0742	-98865.9581	1.0000	19	85.5305	0.0083
20	IV:ADZ:AEZ:BDZ	7	9.1736	22960	2961.3241	1.0000	0.89439067	79.5596	-42958.6759	-154048.4593	1.0000	18	83.6013	0.1846
19	IV:AEZ:BDZ	6	9.3206	14560	2771.1512	1.0000	0.83695391	74.4504	-26348.8488	-96796.0285	1.0000	15	80.2787	0.1846
18	IV:ADZ:BDZ:EZ	6	9.3354	16660	2752.1003	1.0000	0.83120009	73.9386	-30567.8997	-111175.7303	1.0000	15	80.9218	0.1846
17	IV:ABZ:BDZ:EZ	6	9.3672	18760	2710.9844	1.0000	0.81878208	72.8339	-34809.0156	-125577.4972	1.0000	15	79.5284	0.1846
16	IV:BDZ:CZ:EZ	5	9.6061	8176	2401.9813	1.0000	0.72545579	64.5322	-13950.0187	-53508.8196	0.4457	13	74.2765	0.1859
15	IV:AZ:BDZ:EZ	5	9.6284	8260	2373.1234	1.0000	0.71673998	63.7569	-14146.8766	-54112.1036	1.0000	13	73.5263	0.1846
14	IV:BEZ:CZ:DZ	5	9.6329	6426	2367.3323	1.0000	0.71499096	63.6013	-10484.6677	-41576.2595	0.0082	11	73.8478	0.1859
13	IV:BDZ:EZ	4	9.8681	7840	2063.1064	1.0000	0.62310746	55.4279	-13616.8936	-51549.9903	1.0000	8	66.2379	3.4455
12	IV:BEZ:DZ	4	9.8867	6090	2039.0502	1.0000	0.61584190	54.7816	-10140.9498	-39606.8375	0.0000	10	65.8092	3.4455
11	IV:BEZ:CZ	4	9.8956	6146	2027.5271	1.0000	0.61236166	54.4720	-10264.4729	-40001.3112	0.0507	10	63.5584	3.0192
10	IV:BEZ	3	10.1890	5810	1647.9479	1.0000	0.49771965	44.2742	-9972.0521	-38083.1863	1.0000	6	52.5188	35.8173
9*	IV:CZ:DZ:EZ	3	10.2049	826	1627.3872	0.0000	0.49150984	43.7218	-24.6128	-4021.1355	0.0000	5	61.0932	2.8452
8*	IV:BZ:DZ:EZ	3	10.2497	840	1569.5616	0.0000	0.47404514	42.1682	-110.4384	-4174.6987	0.0000	6	58.4137	3.4455
7*	IV:DZ:EZ	2	10.5415	490	1192.0810	0.0000	0.36003697	32.0267	212.0810	-2158.7375	0.0000	3	48.3387	28.2738
6*	IV:BZ:EZ	2	10.5772	560	1145.9266	0.0000	0.34609723	30.7867	25.9266	-2683.5803	0.0026	4	45.3376	35.8173
5*	IV:CZ:EZ	2	10.5901	546	1129.1971	0.0000	0.34104451	30.3373	37.1971	-2604.5722	0.0000	2	46.5166	31.0000
4*	IV:EZ	1	10.9081	210	717.9345	0.0000	0.21683339	19.2882	297.9345	-718.1306	0.0000	1	36.5488	100.0000
3*	IV:DZ	1	11.0063	280	590.9536	0.0000	0.17848212	15.8767	30.9536	-1323.7998	0.0000	1	38.3708	100.0000
2*	IV:CZ	1	11.0118	336	583.8542	0.0000	0.17633793	15.6860	-88.1458	-1713.8499	0.0000	1	37.6206	100.0000
1*	IV:Z	0	11.4632	0	0.0000	1.0000	0.00000000	0.0000	0.0000	0.0000	0.0000	0	28.4030	100.0000
ID	MODEL	Level	H	dDF	dLR	Alpha	Inf	%dH(DV)	dAIC	dBIC	Inc.Alpha	Prog.	%C(Data)	%cover
Best Model(s) by dBIC:														
1*	IV:Z	0	11.4632	0	0.0000	1.0000	0.00000000	0.0000	0.0000	0.0000	0.0000	0	28.4030	100.0000
Best Model(s) by dAIC:														
4*	IV:EZ	1	10.9081	210	717.9345	0.0000	0.21683339	19.2882	297.9345	-718.1306	0.0000	1	36.5488	100.0000
Best Model(s) by Information, with all Inc. Alpha < 0.05:														
9*	IV:CZ:DZ:EZ	3	10.2049	826	1627.3872	0.0000	0.49150984	43.7218	-24.6128	-4021.1355	0.0000	5	61.0932	2.8452

From this table, the best model by information is CZ:DZ:EZ! This is the point just before alpha flips to 1.0 from 0.0. Also, this precedes a huge jump in dDF, which strikes a balance between simplicity and power. Which means that only three previous notes need to be tracked to reasonably predict the next note, not all five.

I take this to heart and run a fit on a series of three note predictions AZ:BZ:CZ. The fit table is too large to print here, but can be found in the fuga repository under `occam/fit/*`. This shows all the combinations of probabilities for each independent variable. These all have a cardinality somewhere between 16 and 30, depending on how many relative jumps were really observed. The dependent variable for this run has 15 observed states.

The next step is parsing these fit tables into trees whose path for each row is given by the states of all the independent variables and whose value at that node is all of the possible outcomes and their relative probabilities. If nothing is found for that path, a shorter path is used on a similar tree created from data with only two previous notes (and therefore TWO independent variables). If this again is not found, a third tree built on the data for a single previous note is used, which is always found. Then, given a seed path

(which can even be a single note), this process can be iterated, casting the three previous notes relative to the latest note and using this as a path into the probability tree, where the resulting state is found from choosing from the possible outcomes weighted by their relative probabilities. This gives an arbitrarily long sequence of notes, which can then be recast into note events and passed on to the midi output player.

Results

Discussion

Conclusion