Music Generation with Relation Join

Xiuyan Ni, Ligon Liu, and Robert Haralick

The Graduate Center, City University of New York
Computer Science Department
New York, NY, 10016, U.S.A
{xni2,lliu1}@gradcenter.cuny.edu
{rharalick}@gc.cuny.edu
http://gc.cuny.edu/Home

Abstract. Given a data set taken over a population, the question of how can we construct possible causal explanatory models for the interactions and dependencies in the population is a causal discovery question. Projection and Relation Join is a way of addressing this question in a non-deterministic context with mathematical relations. In this paper, we apply projection and relation join to music harmonic sequences to generate new sequences in given composers styles. Instead of first learning the patterns, and then making replications as early music generation work did, we introduce a completely new data driven methodology to generate music.

Keywords: music generation, projection, relation join

1 Introduction

Could a computer compose music sequences that are indistinguishable from the work of human composers to average human listeners? Different models have been applied by researchers trying to answer this question.

Early work in music generation use pattern matching process to identify different styles of music. The pattern matching process first designs a pattern matcher to locate the patterns inherited in input works, stores the patterns in a dictionary, then makes replications according to the patterns[1]. Cope's Experiments in Musical Intelligence incorporates the idea of recombinancy, he breaks music pieces into small parts and then recombines them under certain music constraints to generate new music in a given style. The music constraints are learned using augmented transition network (ATN). ATN is a type of graph theoretic structure widely used in Natural Language Processing to parse complex natural language and generate new sentences[2, 3]. The pattern matching algorithm used by Cope matches intervals instead of pitch, that is, (C, E, G) can be matched to $(D, F\#, A)$, or any major triads[2]. Manaris et al. (2007)[4] employ genetic programming to music generation, which uses artificial music critics as fitness functions[5, 6]. Walter and Merwe (2010) use Markov Chains (MCs) and Hidden Markov Models (HMM)[7] to generate music. They use a certain style of music as training data, then apply the LearnPSA algorithm[8] to produce a prediction suffix tree to find all strings with a statistical significance. The relation between a hidden and an observed

sequence is then modeled by an HMM. After the whole learning process, they sample from the distributions learned to generate new music sequences in the same style as the training data[7]. An HMM is also used to classify folk music from different countries[9]. These sampling methods, however, have drawbacks that they may be stuck in local optimal.

Some other music generation methods do not use music pieces as input. They generate music based on certain rules either from music theory, or principles from artificial intelligence algorithms. Ebcioglu (1986)[10] codes music rules in certain styles in formal grammar to generate a specific style of music, which means musical knowledge related to specific style is needed to make new music. Al-Rifaie (2015)[11] applies Stochastic Diffusion Search (SDS), a swarm intelligence algorithm, to new music generation. This method generates music based on input plain text and the interaction between the algorithm and its agents. It maps each letter or pair of letters in a sentence to the MIDI number of a music note, and then calculates the pitch, the note duration and the volume of music notes based on parameters of SDS. The output music does not have any specific style.

In general, the previous music generation methods either depend on music knowledge, or use machine learning techniques that have to estimate the probability of a music sequence. In the second case, they have to learn the probability of one element given previous elements in a sequence. In this paper, we use a completely different methodology to generate music specific to a certain composer. We break each piece in our music corpus into overlapping small segments and use relation join to generate synthetic musical sequences. The relation join replaces the probabilities in methods like MCs, and HMMs.

Take the MCs method as an example to compare a probability based method to our method. A first order MC assumes that $P(x_t|x_{t-1}, \ldots, x_1) = P(x_t|x_{x-1})$, where $< x_1, x_2, \ldots x_t >$ is a sequence of states (a state can be a chord or a note). It estimates those probabilities given a music corpus and then generates music sequences based on the probabilities. While in our method, we first break the music sequences into small segments according to specific length and number of overlapping notes (chords). Then we reconstruct music sequences using the set of segments. We call each sequence or small part a tuple (See definition 6). For example, we have a tuple sequence $< x_1, x_2, \ldots x_t >$, and we set tuple length of each segment to 4, and overlapping number to 2. We first break the tuple sequence into a set of tuples $\{< x_1, x_2, x_3, x_4 >, < x_3, x_4, x_5, x_6 >, < x_5, x_6, x_7, x_8 >, \ldots\}$. If we repeat the first step for all sequences, we will get a dataset that contain all possible 4-tuples with overlap of 2 (4 consecutive chords) for a given music corpus which contain sequences of chords. Then we generate a chord sequence by randomly selecting one 4-tuple from the set that contains all 4-tuples from the music corpus, then look at the last two chords of the selected tuple, and select another 4-tuple from the subset that contains all 4-tuples starting with the last two chords from previous 4-tuple until we reach a certain length. If the process gets stuck because there is no possible consistent selection, the process backtracks in a tree search manner.

Thus, in our method, there is no need to estimate probabilities. For any 4-tuple (not the first or the last) in a generated sequence, the 4-tuple in the generated sequence are

consistent with the 4-tuple that precedes it and that follows it in the generated music sequence. Our music generation method is like a solution to a constraint satisfaction problem. It can therefore be posed in a rule-based mode as well.

Our method can be used without musical knowledge of different styles, and we do not need to learn patterns or parameters from input music pieces either. We use the idea of recombination (first breaking the input music into small parts, and then recombine them to generate new music sequences), but we don't have to estimate the probabilities. The idea of this method is that the progressions inherent in music sequences carry the patterns of music of different composers themselves.

We will describe this method in detail in Section 2. Section 3 will demonstrate how this method is applied to music generation. Several experiments are introduced in Section 4. Section 5 concludes current work and looks into future work.

2 Definition

In order to introduce the procedure of applying relation join to music sequences, we formally define the concepts used in this section[12].

Definition 1. *Let $X_1, ..., X_N$ be the N variables associated with a relation. Let L_n be the set of possible values variable X_n can take. Let R be a data set or knowledge constraint relation. Then*

$$R \subseteq \bigtimes_{i=1}^{N} L_i \tag{1}$$

We will be working with many relations associated with different and overlapping variable sets and therefore over different domains. For this purpose we will carry an index set along with each relation. The index set indexes the variables associated with the relation. An index set is a totally ordered set.

Definition 2. *$I = \{i_1, ..., i_K\}$ is an index set if and only if $i_1 < i_2 < \cdots < i_K$.*

Next we need to define Cartesian product sets with respect to an index set.

Definition 3. *If $I = \{i_1, ..., i_K\}$ is an index set, we define Cartesian product:*

$$\bigtimes_{i \in I} L_i = \bigtimes_{k=1}^{K} L_{i_k} = L_{i_1} \times L_{i_2} \times ... \times L_{i_K} \tag{2}$$

The definition tells us that the order in which we take the Cartesian product $\bigtimes_{i \in I} L_i$ is precisely the order of the indexes in I.

For a natural number N, we use the convention that $[N] = \{1, ..., N\}$ and $|A|$ designates the number of elements in the set A.

Now we can define the indexed relation as a pair consisting of an index set of a relation and a relation.

Definition 4. *If I is an index set with $|I| = N$ and $R \subseteq \bigtimes_{i\in I} L_i$, then we say (I, R) is an indexed $N-$ary relation on the range sets indexed by I. We also say that (I, R) has dimension N. We take the range sets to be fixed. So to save writing, anytime we have an indexed relation (I, R) , we assume that that $R \subseteq \bigtimes_{i\in I} L_i$, the sets L_i, $i \in I$, being the fixed range sets.*

We will be needing to define one relation in terms of another. For this purpose, we will need a function that relates the indexes associated with one relation to that of another. We call this function the index function.

Definition 5. *Let J and M be index sets with*

- $J = \{j_1, \ldots, j_{|J|}\}$
- $M = \{m_1, \ldots, m_{|M|}\}$
- $J \subset M$

The index function $f_{JM} : [|J|] \rightarrow [|M|]$ is defined by $f_{JM}(p) = q$ where $m_q = j_p$. The index function f_{JM} operates on the place p of an index from the smaller index set and specifies where – place q – in the larger index set that the index j_p can be found; thus $m_q = j_p$.

Another important concepts we need before we define project and relation join is tuple. Also what is a tuple's length.

Definition 6. *A tuple is a finite ordered list of elements. An n-tuple is a sequence (or ordered list) of n elements, where n is a non-negative integer. We call n the length of the n-tuple.*

Next we need the concept of projection since it is used in the definition of relation join. If (J, R) is an indexed relation and $I \subseteq J$, the projection of (J, R) onto the ranges sets indexed by I is the indexed set (I, S) where a tuple $(x_1, ..., x_{|I|})$ is in S whenever for some $|J|$-tuple $(a_1, ..., a_{|J|})$ of R, x_i is the value of that component of $(a_1, ..., a_{|J|})$ in place $f_{IJ}(i)$.

Definition 7. *Let I and J be index sets with $I \subseteq J$. The projection operator projecting a relation on the range sets indexed by J onto the range sets indexed by I is defined by $\pi_I(J, R) = (I, S)$ where*

$$S = \left\{ (x_1, ..., x_I) \in \bigtimes_{i\in I} L_i \mid \exists (a_1, ..., a_{|J|}) \in R,\ a_{f_{IJ}(i)} = x_i,\ i \in I \right\} \tag{3}$$

That is,

$$\pi_I(J, (a_1, ..., a_{|J|})) = \left(I, \left(a_{f_{IJ}(1)}, ..., a_{f_{IJ}(|I|)}\right)\right) \tag{4}$$

If $I \cap J^c \neq \emptyset$, then $\pi_I(J, R) = \emptyset$

The operation of projection is overloaded, and if $R \subseteq \bigtimes_{n=1}^{N} L_n$ and $I \subseteq \{1, \ldots, N\}$, we define

$$\pi_I(R) = \pi_I(\{1, \ldots, N\}, R) \tag{5}$$

A relation join can be thought of as the equijoin or natural join operation in the data base world.

Definition 8. *Let (I,R) and (J,S) be indexed relations, let $K = I \cup J$ and L_k be the range set for variable $k \in K$. Then the relation join of (I,R) and (J,S) is denoted by $(I,R) \otimes (J,S)$, and is defined by*

$$(I,R) \otimes (J,S) = \left\{ t \in \bigtimes_{k \in K} L_k \mid \pi_I(K,t) \in (I,R) \text{ and } \pi_J(K,t) \in (J,R) \right\} \tag{6}$$

Example 1. Following is an example for relation join. If we have two indexed relations, (I,R) and (J,S) as in Table 1, the relation join for the two relations will be as in Table 2.

Table 1. Values for (I,R) and (J,S)

(I,R)		(J,S)	
I	1,4,7,9	J	2,4,6,7
1	(a,b,e,d)	1	(e,e,a,d)
2	(b,d,e,a)	2	(d,c,b,a)
3	(e,c,a,b)	3	(a,d,b,e)
4	(c,e,d,a)	4	(b,b,c,e)

Table 2. Values for (I,R) and (J,S)

	$(K,T) = (I,R) \otimes (J,S)$
K	1,2,4,6,7,9
$(1,4)$	(a,b,b,c,e,d)
$(2,3)$	(b,a,d,b,e,a)
$(3,2)$	(e,d,c,b,a,b)
$(4,1)$	(c,e,e,a,d,a)

3 Music Generation through Projection and Relation Join

Section 2 introduced the definition of project and relation join which are the core techniques we will use in the music generation. In this section, we will introduce how the techniques can be applied to music sequences (chord sequences). Before that, we need to introduce the mathematical definition we use for music terms.

Definition 9. *A note is a small bit of sound with a dominant fundamental to introduce the frequency and harmonics sound. For the sake of simplicity, this domain includes all the notes on a piano keyboard,*

we define a set of notes N as:

$$N = \{A0, B0, C1, C\#1, D1, \ldots, B7, C8\} \tag{7}$$

In music, a chord is a set of notes that is heard sounding simultaneously.

Definition 10. *A chord is a set of notes, that is, for any chord c, $c \subseteq N$.*

Now we can define a music sequence such as an harmonic sequence.

Definition 11. *Let C be a collection of all chords, the harmonic sequence of a musical piece of length L is then a tuple $h \in C^L$.*

A music corpus can be represented as a set H of Z Harmonic Sequences. $H = \left\{h_z \in C^{L_z}\right\}_{z=1}^{Z}$, where L_z is the length of the tuple h_z.

An example of an harmonic sequence with 8 chords is as following:

Example 2. {'B4', 'E4', 'D4', 'G3'}, {'A4', 'E4', 'C#4', 'E3', 'A3'}, {'G4', 'E4', 'C#4', 'E3', 'A3'}, {'A5', 'F#4', 'E4', 'C#4', 'A3', 'A2'}, {'G5', 'E4', 'C#4', 'A3', 'A2'}, {'F#5', 'F#4', 'D4', 'A3', 'D3'}, {'E5', 'F#4', 'D4', 'A3', 'D3'}, {'D5', 'F#4', 'D4', 'A3', 'D3', 'F#3'}

We know that there exist certain rules in chords progressions to make a harmonic sequences sound consistent. To take advantages of those rules, we need to design index sets for the sequences to project on.

Definition 12. *A collection $\mathcal{I}(m, n)$ of K length m sets with uniform overlap of n ($n < m$) is represented as:*

$$\mathcal{I}(m, n) = \{I_k \mid I_k = \{(m-n)\cdot k + 1, (m-n)\cdot k + 2, \ldots, (m-n)\cdot k + m\}\}_{k=0}^{K-1} \tag{8}$$

$\mathcal{I}(m, n)$ is a collection of tuple sets.

For example, if $m = 4$ and $n = 2$, the tuple sets are:

Example 3.

$$I_0 = \{1, 2, 3, 4\}$$

$$I_1 = \{3, 4, 5, 6\}$$

$$I_2 = \{5, 6, 7, 8\}$$

$$\vdots$$

$$I_{K-1} = 2\cdot(K-1) + 1, 2\cdot(K-1) + 2, 2\cdot(K-1) + 3, 2\cdot(K-1) + 4$$

$\mathcal{I}(4, 2) = \{I_0, I_1, \ldots, I_{K-1}\}$.

We can now collect data sets from music sequences based on the tuple sets.

Theorem 1. *Let $\mathcal{I} = \mathcal{I}(m, n)$ be a collection of K length m sets with uniform overlap of n, let h be a harmonic sequence, the set R_h of all m-tuples with overlap n from h is defined by*

$$([(m-n)\cdot(K-1)+m], R_h) = \cup_{I\in\mathcal{I}}\pi_I(h) \tag{9}$$

If H is set of harmonic sequences, then

$$([(m-n)\cdot(K-1)+m], R) = \cup_{h\in H}\cup_{I\in\mathcal{I}}\pi_I(h) \tag{10}$$

As an example,

Example 4. If we have two pieces. The first piece is: <{'B4', 'E4', 'D4', 'G3'}, {'A4', 'E4', 'C#4', 'E3', 'A3'}, {'G4', 'E4', 'C#4', 'E3', 'A3'}, {'A5', 'F#4', 'E4', 'C#4', 'A3', 'A2'}, {'G5', 'E4', 'C#4', 'A3', 'A2'}, {'F#5', 'F#4', 'D4', 'A3', 'D3'}, {'E5', 'F#4', 'D4', 'A3', 'D3'}, {'D5', 'F#4', 'D4', 'A3', 'D3', 'F#3'} >, as in the sheet shown in Fig.1. The second piece is: <{'D5', 'E4', 'D4', 'B3', 'G3', 'G2'}, {'C#5', 'E4', 'D4',

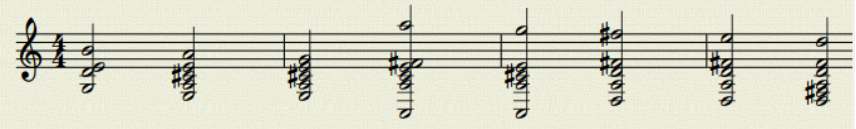

Fig. 1. The First Example of 8-Tuple

'G3'}, {'B4', 'E4', 'D4', 'G3'}, {'A4', 'E4', 'C#4', 'E3', 'A3'}, {'G4', 'E4', 'C#4', 'E3', 'A3'}, {'A5', 'F#4', 'E4', 'C#4', 'A3', 'A2'}, {'G5', 'E4', 'C#4', 'A3', 'A2'}, {'F#5', 'F#4', 'D4', 'A3', 'D3'} >, as in the sheet shown in Fig.2.

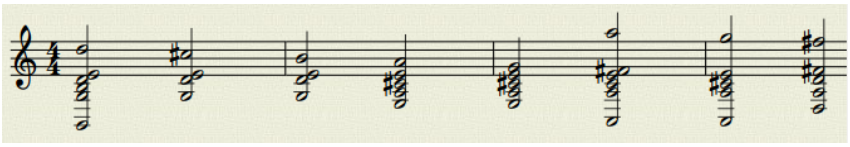

Fig. 2. The Second Example of 8-Tuple

If $m = 4$ and $n = 2$, five 4-tuple will be generated. R = { <{'B4','E4', 'D4', 'G3'}, {'A4', 'E4', 'C#4', 'E3', 'A3'}, {'G4', 'E4','C#4', 'E3', 'A3'}, {'A5', 'F#4', 'E4', 'C#4', 'A3', 'A2'}>, <{'G4', 'E4', 'C#4', 'E3', 'A3'}, {'A5', 'F#4', 'E4', 'C#4','A3', 'A2'}, {'G5', 'E4', 'C#4', 'A3', 'A2'}, {'F#5', 'F#4','D4', 'A3', 'D3'}>, <{'G5', 'E4', 'C#4', 'A3', 'A2'}, {'F#5', 'F#4', 'D4', 'A3', 'D3'}, {'E5', 'F#4', 'D4', 'A3', 'D3'}, {'D5','F#4', 'D4', 'A3', 'D3', 'F#3'}>, <{'D5', 'E4', 'D4', 'B3', 'G3','G2'}, {'C#5', 'E4', 'D4', 'G3'}, {'B4', 'E4', 'D4', 'G3'},{'A4', 'E4', 'C#4', 'E3', 'A3'}>, <{'G4',

'E4', 'C#4', 'E3', 'A3'}, {'A5', 'F#4', 'E4', 'C#4', 'A3', 'A2'}, {'G5', 'E4', 'C#4', 'A3', 'A2'}, {'F#5', 'F#4', 'D4', 'A3', 'D3'}>}

Now we can define the relation join for harmonic sequences.

Definition 13. *If R is a set of m-tuples produced from projections with index set $\mathcal{I} = \mathcal{I}(m, n)$, and if $I \in \mathcal{I}$ is an index set, (I, R) becomes an indexed relation. Let $J = \cup_{I \in \mathcal{I}} I$, we then can get new harmonic sequences by computing*

$$(J, S) = \otimes_{I \in \mathcal{I}} (I, R) \tag{11}$$

The above procedure can be applied to harmonic sequences with and without corresponding time duration. But there is no intentional control of key of harmonic sequences in this procedure.

Definition 14. *Let K be the set of all possible keys in music, then*

$$K = \{C, Db; D; Eb; E; F; Gb; G; Ab; A; Bb, B\} \tag{12}$$

Enharmonic keys are counted as one key, that is, $C\# = Db; D\# = Eb; F\# = Gb; G\# = Ab; A\# = Bb; Cb = B$

When we say a piece is in a certain 'key', it means the piece is formed around the notes in a certain scale which, in music, is a set of notes ordered by certain frequency or pitch. For example, the C Major Scale contains C, D, E, F, G, A, B, and C. A piece based on the key of C will (generally) use C, D, E, F, G, A, B, and C.

Now we could do **key constraint relation join**.

Definition 15. *Let*

$$R_k^b = \{(c_1, c_2, \ldots, c_m) \mid c_1 \in C_k\} \tag{13}$$

$$R_k^e = \{(c_1, c_2, \ldots, c_m) \mid c_m \in C_k\} \tag{14}$$

where C_k is a set of chords who are in the key of k, $R_k^b \subseteq R$ contains all m-tuples of chords in which the first chord is in the key of k, $'b'$ means begin. Similarly, $R_k^e \subseteq R$ contains all m-tuples of chords in which the last chord is in the key of k, $'e'$ means end.

Then compute

$$(J, S) = \left(I_0, R_k^b\right) \otimes_{i=1}^{K-2} (I_i, R) \otimes \left(I_{K-1}, R_k^e\right) \tag{15}$$

Which is relation join constrained by using chords in the key of k that begin and end the piece.

We could also do **scale constraint relation join**.

Definition 16. *A scale, in music, is a set of notes ordered by certain frequency or pitch. For example, the C Major Scale contains C, D, E, F, G, A, B, and C.*

Let $R_S \subseteq R$ be a set of tuples of chords in which all chords are in scale S. Then we can get new harmonic sequences in which all chords are in scale S by computing

$$(J, S) = \otimes_{I \in \mathcal{I}} (I, R_S) \tag{16}$$

4 Experiments

In this section, we apply the techniques introduced in Section 2 and 3 to a music corpus from Music21[1].

4.1 Experiment 1: Harmonic Sequence

There are five steps in this experiment.

Firstly, extract chords. We extract chords from 202 music sequences of Bach from the database of Music21. Every sample is a list including several tuples. Every tuple represents a chord, which contains all the notes in the chord. As an example,

$$< \{'F4','C4','A3','F3'\}, \{'G4','C5','C4','G3','E3'\}, \{'C4','C5','G3','E3'\}, ... >$$

is a harmonic sequence sample.

Secondly, transform chords into integer indexes. We make a dictionary(mapping) for all the chords, the key of the dictionary is each chord itself, the value is the integer index from index set $\{0, 1, 2, ...D-1\}$, where D is the number of distinct chords. Then, transform the chords in each sample into the integer indexes according to the dictionary.

Thirdly, get all tuples of chord from music piece samples. In this experiment, we set $\mathcal{I} = \mathcal{I}(4, 2)$, that is, $m = 4$, $n = 2$, $K = 14$[2], then compute

$$([(m-n)\cdot(K-1)+m], R) = \cup_{h\in H} \cup_{I\in\mathcal{I}} \pi_I(h) \tag{17}$$

There are 8731 4-tuples extracted from the music sequences in this experiment.

Fourthly, do relation join on the projected index relations, that is, compute

$$(J, S) = \otimes_{I\in\mathcal{I}}(I, R) \tag{18}$$

Fifthly, create mp3 files from the new pieces generated in the fifth step. There are two sub-steps in this step: first, swap the keys and values of dictionary $dic1$ to get a new dictionary $dic2$, that is, $dic2$ is the inverse mapping of $dic1$; second, transform the new chords sequences represented by index into chords lists according to $dic2$, and generate mp3 files from the new chords lists.

The relation join procedure, if done completely, generates over 24.12 million harmonic sequences in this experiment. We pick samples using a tree search method. We randomly pick one tuple from R, and then pick the next tuple that can join onto it. If there are no tuples can join onto it, then the procedure backtracks in a tree search manner. In this way, we can get certain number of synthetically generated sequences. Another way to pick the sample is to randomly select from the results of a full relation join. This can be very time consuming, because we need to get all the results before sampling. After we have some samples, we can make them into mp3 files that can be listened to.

[1] Music 21 is a toolkit for computer-aided musicology. See http://web.mit.edu/music21/.

[2] K is set to 14 to ensure the length of each output sample is 32, which is a reasonable length of a harmonic sequence sample.

4.2 Experiment 2: Harmonic Sequence with Rhythm

In this experiment, instead of only extracting information of the chords, we include the information of rhythm for each chord. Thus, each chord comes with its time duration. There are 8773 4-tuples extracted in the third step in this experiment.

In the first step, we extract a harmonic sequence sample as following, the number at the end of each chord is the time duration in quarter length, 1.0 represents a quarter, 0.5 represents a eighth, and so on:

$$< \{'F4',' C4',' A3',' F3', 0.5\}, \{'G4',' C5',' C4',' G3',' E3', 0.5\}, \{'C4',' C5',' G3',' E3', 1.0\}, ... >$$

The other four steps are the same as in experiment 1. Relation join generates above 1.67 million sequences in this experiment.

4.3 Experiment 3: Harmonic Sequence in Specific Key and Scale

In the above experiments, there is no intentional control of the key of harmonic sequences and the scale the chords in. We want to see if the harmonic sequences sound better when we specify the key and scale. So we do two constraint relation join experiments based on each of the above two experiments, which will generate four combinations of experiments. The number of harmonic sequences each experiment generated are summarized in table 3.

Table 3. The number of sequences generated with key and scale constraint

type	with key constraint	with scale constraint
chord	65648	577602
chord with rhythm	4958	867977

Since relation join generates new sequences using existing harmonic sequences, it relies on the transitions of chords of existing sequences. In addition, machine generated sequences will have the same length, while the human generated sequences have more sequential features of longer length.

4.4 Experiment 4: Redo the Experiments with $m = 5$, $n = 3$

We also do another set of experiments with $m = 5$, $n = 3$. We extract 8797 and 8813 5-tuples from the 202 Music21 sequences respectively for tuples with only chord and tuples including both chord and rhythm. The results are summarized in Table 4.

Some samples from these experiments are also posted to the website: `http://haralick.org/music/music.html`.

[3] Except this experiment, the time duration of all chords with rhythms are restricted to be half chord

Table 4. The number of sequences generated with key and scale constraint

type	no constraints	with key constraint	with scale constraint
chord	63262	266	365
chord with rhythm	571	119 [3]	1

5 Conclusion and Future Work

Previous music generation methods try to find music sequences set:

$$\{x_1, x_2, \ldots, x_N | P(x_1, x_2, \ldots, x_N) > 0\} \tag{19}$$

they use machine learning techniques to estimate:

$$P(x_1, x_2, \ldots, x_N) = \prod_{k=1}^{K} f_k(x_i : i \in A_k) \tag{20}$$

for all $x_1, \ldots x_N \in \bigtimes_{i=1}^{N} L_i$, where L_i are the space of music elements (such as chords), N is the length of each sequence, A_k is a set of index tuples.

Only those music sequences with $P(x_1, x_2, \ldots, x_N) > 0$ will be generated. So they have to estimate $f_k(x_i : i \in A_k)$ and make sure that for $\forall k$, $f_k(x_i : i \in A_k) > 0$.

In this paper, we use a completely different methodology to generate music specific to certain composers called projection and relation join. Instead of estimating the probabilities, we calculate the relation join of (A_k, R_k) for all k,

in which

$$(A_k, R_k) = (A_k, \{(x_i, i \in A_k) | f_k(x_i, i \in A_k) > 0\}) \tag{21}$$

Thus, in our method, $f_k(x_i : i \in A_k) > 0$ is ensured for any k.

This method requires neither expert level domain knowledge nor learning patterns and parameters from input music pieces. The method is based on the idea of recombination, but without estimating any probabilities. The idea of this method is that the progressions inherent in music sequences themselves carry on the patterns of music of different composers.

References

1. Papadopoulos,G.,Wiggins,G.: AI Methods for Algorithmic Composition: A survey, a Critical View and Future Prospects. In: AISB Symposium on Musical Creativity, Edinburgh, UK, 110-117 (1999)
2. Cope, D.: Computer Modeling of Musical Intelligence in EMI. Computer Music Journal, 69-83 (1992)
3. Winograd, T.: Language As a Cognitive Process: Volume 1: Syntax. (1983)
4. Manaris, B., Roos, P., Machado, P., Krehbiel, D., Pellicoro, L. and Romero, J.: A Corpus-based Hybrid Approach to Music Analysis and Composition. In Proceedings of the National Conference on Artificial Intelligence (Vol. 22, No. 1, p. 839). Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2007)

5. Romero, J., Machado, P., Santos, A., Cardoso, A.: On the Development of Critics in Evolutionary Computation Artists. In: Applications of Evolutionary Computing, 559-569, Springer (2003)
6. Machado, P., Romero, J., Manaris, B.: Experiments in Computational Aesthetics. In: The Art of Artificial Evolution, 381-415, Springer (2008)
7. Schulze, W., Van der Merwe, B.: Music Generation with Markov Models. IEEE MultiMedia (3) 78-85 (2010)
8. Ron,D.,Singer,Y.,Tishby,N.: The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. Machine learning 25(2-3), 117-149 (1996)
9. Chai,W.,Vercoe,B.: Folk Music Classification Using Hidden Markov Models.In:Proceedings of International Conference on Artificial Intelligence. Volume 6., Citeseer (2001)
10. Ebcioglu, K.: An Expert System for Harmonization of Chorales in the Style of JS Bach. (1986)
11. Al-Rifaie, A.M., Al-Rifaie, M.M.: Generative Music with Stochastic Diffusion Search. In: Evolutionary and Biologically Inspired Music, Sound, Art and Design, 1-14, Springer (2015)
12. Haralick, R.M., Liu, L., Misshula, E.: Relation Decomposition: the Theory. In: Machine Learning and Data Mining in Pattern Recognition, 311-324, Springer (2013)