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 explanatory models for the interactions and dependencies in the population is a 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 a given composer or genre style. 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. Then we discuss exploring the difference between the original music and synthetic music sequences using information theory based techniques.

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 a pattern matching process to identify different styles of music. The pattern matching process first designs a pattern matcher to locate the patterns inherent in the input music corpus, stores the patterns in a dictionary, and 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 segments and then recombines them under certain music constraints to generate new music in a given style. The music constraints are learned using an 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 of the 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 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 get 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 a 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 a previous number of elements in a sequence. In this paper, we use a completely different methodology to generate music specific to a certain composer or genre. 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 Markov Chains, and HMMs.

Consider the Markov Chain method as an example to compare a probability based method to our method. A first order Markov Chain assumes that $P(x_t|x_{t-1}, \ldots, x_1) = P(x_t|x_{t-1})$, where $< x_1, x_2, \ldots x_t >$ is a sequence of states (a state can be a chord or a note). Changes in state are called transitions. The Markov model estimates those probabilities of transitions given a music corpus and then generates music sequences based on the probabilities by linking up the transitions. While in our method, we first break the music sequences into small segments of given length and number of overlapping notes (chords). Then we reconstruct music sequences using the set of segments.

We call each sequence or segment of musical chords a tuple (See definition 5). For example, we have a tuple sequence $< x_1, x_2, \ldots x_t >$, and we can set the tuple length of each segment to 4 consecutive chords, and overlapping number to 2. We first break the tuple sequence into a set of tuples $\{< x_1, x_2, x_3, x_4 >, < x_2, x_3, x_4, x_5 >, < x_3, x_4, x_5, x_6 >, < x_5, x_6, x_7, x_8 >, \ldots\}$. If we repeat the first step for all sequences, we will get a dataset that contains all possible 4-tuples with overlap of 2 for a given music corpus which contains 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 depth first 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 is 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[21]. 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 music theoretic patterns or parameters from input music pieces either. We use the idea of recombination (first breaking the input music into small segments, and then recombine them to generate new music sequences), but we do not have to estimate the probabilities. The idea of this method is that the progressions inherent in music sequences carry the composer patterns themselves.

We then discuss a technique to distinguish between original music and synthetic music. We use the information based distance based on mutual information and entropy of random variables, and visualize the results the auto similarity function for both original and synthetic music sequences.

We describe our generation 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 our current work and discusses our future work which will apply an information based distance to compare original music and synthetic music.

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*

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

Example 1. When $N = 2$, and $L_1 = \{a, b\}$, and $L_2 = \{c, d, e\}$, R will be a subset of $\bigtimes_{i=1}^{N} L_i = \{(a, c), (a, d), (a, e), (b, c), (b, d), (b, e)\}$.

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:*

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

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 $R \subseteq \bigtimes_{i\in I} L_i$, the sets L_i, $i \in I$, being the fixed range sets.*

Example 2. We continue the example in Example. 1 to illustrate this definition. Let $I = \{7, 9\}$, the L_1 and L_2 are the same as in Example 1. We know that R is a subset of $\bigtimes_{i=1}^{N} L_i = \{(a, c), (a, d), (a, e), (b, c), (b, d), (b, e)\}$. Let $R = \{(a, c), (a, d), (a, e), (b, c)\}$, then (I, R) will be an indexed relation as in Table 1.

Another important concept we need before we define projection and relation join is tuple and tuple length.

Definition 5. *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.*

Table 1: *Indexed relation (I, R) where $I = \{7, 9\}$ and $R = \{(a, c), (a, d), (a, e), (b, c)\}$. The 1,2,3,4 are indexes for the row in which the tuple appears. We will make use of the row index to help illustrate how relation join works.*

	(I, R)
$I = \{7, 9\}$	
1	(a, c)
2	(a, d)
3	(a, e)
4	(b, c)

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 6. *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 $f_{JM} : [|J|] \to [|M|]$ 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 $m_q = j_p$. To make this concrete consider the following example. Let I, J and M be index sets with $I \subseteq J \subseteq M$, where $I = \{i_1, i_2\} = \{5, 8\}$, $J = \{j_1, j_2, j_3, j_4\} = \{2, 5, 8, 9\}$ and $M = \{m_1, m_2, m_3, m_4, m_5, m_6\} = \{2, 4, 5, 6, 7, 8, 9\}$. Then, f_{IJ}, f_{JM}, and f_{IM} are defined as in Table 2. The first value in I is i_1. It has the value 5. The place where 5 occurs in the J is the second place. Therefore, $f_{IJ}(1) = 2$, and $j_{f_{IJ}(1)} = 5$. The place where 5 occurs in M is the third place. Therefore, $f_{IM}(1) = 3$ and $m_{f_{IM}(1)} = 5$.

Table 2: *Shows example index sets and the index functions that relate one set to another.*

p	i_p	$f_{IJ}(p)$	$j_{f_{IJ}(p)}$
1	5	2	5
2	8	3	8

p	j_p	$f_{JM}(p)$	$m_{f_{JM}(p)}$
1	2	1	2
2	5	3	5
3	8	6	8
4	9	7	9

p	i_p	$f_{IM}(p)$	$m_{f_{IM}(p)}$
1	5	3	5
2	8	6	8

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*

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

That is,

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

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

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

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

Example 3. Use the example in Example. 2, and let $I' = \{7\}$, $\pi_I(R)$ will be as in Table 3.

Table 3: *Projection of* (I, R) *where* $I = \{7, 9\}$ *and* $R = \{(a, c), (a, d), (a, e), (b, c)\}$ *on* $I' = \{7\}$.

	$\pi_I(R)$
$I' =$	{7}
1	a
2	b

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*

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

An example of relation join is given in Example. 4.

Example 4. Shows an example for relation join. If we have two indexed relations, (I, R) and (J, S) as in Table 4, the relation join for the two relations will be as in Table 5.

Table 4: *Values for indexed relation* (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)

Definition 9. *Let* (I, R) *be an indexed relation with* $R \subseteq \bigtimes_{i \in I} L_i$. *Let* $J \subseteq I$ *and* $a \in \bigtimes_{j \in J} L_j$. *Then the restriction of* (I, R) *to* (J, a) *is denoted by* $(I, R)|_{(J,a)}$ *and is defined by*

$$(I, R)|_{(J,a)} = (I, \{r \in R \mid \pi_J(I, r) = (J, a)\})$$

Example 5. If we have an indexed relation (I, R), where $I = \{1, 2\}$, $R = \{(1, m), (1, n), (2, m), (3, p), (4, q), (5, p)\}$, $J = \{2\}$, and $a = m$, $(I, R)|_{(J,a)} = \{(1, m), (2, m)\}$. □

Table 5: *Shows the relation join results of indexed relation* (I, R) *and* (J, S)*. The first column of* K *indicates which row rows are joined from Table. 4. The first number is the row number of the relation* (I, R) *and the second number is the row number of the relation* (J, S)*. For example,* $(1, 4)$ *means the first row of* (I, R) *and the fourth row of* (J, S) *can be joined. However, the first row of* (I, R) *and the first row of* (J, S) *can not be joined because for index 4 of* I*, the component of the tuple on the first row of* (I, R) *has the value* b*. But for index 4 of* J*, the component on the first row of* (J, S) *has the value* e*.*

	$(K, T) = (I, R) \otimes (J, S)$
	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)

Definition 10. *Let* (K, R) *be an indexed relation and* $\{I, J, M\}$ *a non-trivial partition of* K*. We say conditioned on* M*,* I *has no influence on* J *if and only if*

$$\cup_{(M,c)\in\pi_M(K,R)}\pi_I((K,R)|_{(M,c)}) \otimes (M, c) \otimes \pi_J((K,R)|_{(M,c)}) \subseteq (K, R)$$

If $\{I, J\}$ *constitutes a partition of* K*, then we say* I *has no influence on* J *if and only if*

$$\pi_I(K, R) \otimes \pi_J(K, R) \subseteq (K, R)$$

The no influence concept for relations is analogous to the conditional independence concept in probability distributions. For example, in a Markov chain $< x_1, \ldots, x_N >$ of order 1, x_{i+2} is conditionally independent of x_i given x_{i+1}, $i = 2, \ldots, N-1$. In the example of Figure 1, 3 has no influence on 1 given 2. Here we use the index i rather than the variable x_i to simplify our writing.

Figure 1 shows an example of the no influence concept. 1.

$$(I,R)\otimes(J,S)=\bigcup_{(M,c)\in\pi_M(I,R)\cap\pi_M(J,S)}\pi_{I-M}(I,R)|_{(M,c)}\otimes(M,c)\otimes\pi_{J-M}(J,S)|_{(M,c)}$$

(I,R)	
1	**2**
1	a
1	b
2	a
3	c
4	b
5	d

(J,S)	
2	**3**
a	α
a	β
a	γ
b	α
b	β
b	δ
c	ϵ
c	λ
e	ϵ

$(I,R)\otimes(J,S)$		
1	**2**	**3**
1	a	α
1	a	β
1	a	γ
2	a	α
2	a	β
2	a	γ
1	b	α
1	b	β
1	b	δ
4	b	α
4	b	β
4	b	δ
3	c	ϵ
3	c	λ

$$\begin{aligned}
I &= \{1,2\}\\
J &= \{2,3\}\\
M &= I\cap J=\{2\}\\
I-M &= \{1\}\\
J-M &= \{3\}\\
\pi_2(I,R) &= \{a,b,c,d\}\\
\pi_2(J,S) &= \{a,b,c,e\}\\
\pi_2(I,R)\cap\pi_2(J,S) &= \{a,b,c\}
\end{aligned}$$

Fig. 1: *From the above table, we can see that (M,c), the value corresponding to index M in a tuple, does not give us any information about the relations between $\pi_{I-M}(I,R)$ and $\pi_{J-M}(J,S)$, since given a value corresponding to any index in M, all possible values corresponding to any index of $I-M$ pair with all possible values of any index of $J-M$. That is, $I-M$ has no influence on $J-M$ given (M,c).*

The relation join process also applies to the case when we have more than two relations to join. If we have N indexed relations to join, $(I_1,R_1),\ldots,(I_n,R_n)$, we write

$$(K,S)=\otimes_{n=1}^{N}(I_n,R_n).$$

3 Music Generation through Projection and Relation Join

Section 2 introduced the definition of projection 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 (note or chord sequences). Before that, we need to introduce the mathematical definition we use for music terms.

Definition 11. *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:*

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

The number after each note represents the octave the note in. The frequency of one note is the frequency of the previous note multiplied to $\sqrt[12]{2}$. Figure 2 shows the note name versus the frequency of the twelve notes in the octave Middle C in.

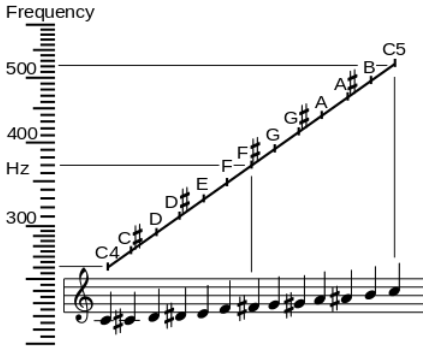


Fig. 2: *Each note shown has a frequency of the previous note multiplied by* $\sqrt[12]{2}$. *Image from* $https://en.wikipedia.org/wiki/Musical\_note$.

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

Definition 12. *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.

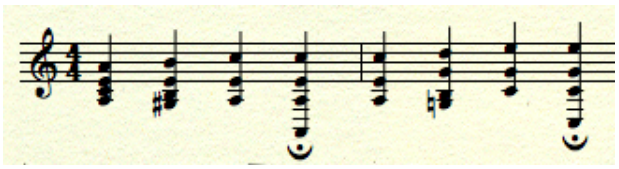
Definition 13. *Let C be a collection of all chords, the harmonic sequence of a musical piece of length L is then a tuple* $h \in C^L$, *where* $C^L = \underbrace{C \times C \times \ldots \times C}_{L \text{ times}}$

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 in Figure 3.

<{'A4', 'E4', 'C4', 'A3'}, {'B4', 'E4', 'B3', 'G#3'}, {'C5', 'E4', 'A3'}, {'C5', 'E4', 'A3', 'A2'}, {'C5', 'E4', 'A3'}, {'D5', 'G4', 'B3', 'G3'}, {'E5', 'G4', 'C4'}, {'E5', 'G4', 'C4', 'C3'} >

(a) *The harmonic sequence written as in tuple form*



(b) *The harmonic sequence written in a sheet music*

Fig. 3: *A harmonic sequence with 8 chords*

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

Definition 14. *A collection $\mathcal{I}(m,n)$ of K length m sets with uniform overlap of n $(0 < n < m)$ is represented as:*

$$\mathcal{I}(m,n) = \bigcup_{i\in\{1,\ldots,n-1\}} \mathcal{I}^i(m,n),$$

where

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

If $m = 4$ and $n = 2$, and $i = 0$, the tuple sets are shown in Figure 4. When m and n are calculated, we suppress the (m,n), and write $\mathcal{I} : \mathcal{I} = \mathcal{I}(m,n)$.

$$I_0^0 = \{1,2,3,4\}$$
$$I_1^0 = \{3,4,5,6\}$$
$$I_2^0 = \{5,6,7,8\}$$
$$\vdots$$
$$I_{K-1}^0 = 2\cdot(K-1)+1, 2\cdot(K-1)+2, 2\cdot(K-1)+3, 2\cdot(K-1)+4$$

(a) *The tuple sets when $i = 0$*

$$I_0^1 = \{2,3,4,5\}$$
$$I_1^1 = \{4,5,6,7\}$$
$$I_2^1 = \{6,7,8,9\}$$
$$\vdots$$
$$I_{K-1}^1 = 2\cdot(K-1)+1-1, 2\cdot(K-1)+2-1, 2\cdot(K-1)+3-1, 2\cdot(K-1)+4-1$$

(b) *The tuple sets with $i = 1$*

Fig. 4: *Shows the sample tuple sets when $m = 4$, and $n = 2$. All tuple sets are as in* $\mathcal{I}(4,2) = \left\{ I_0^0, I_1^0, \ldots, I_{K-1}^0, \ldots, I_0^{K-1}, I_1^{K-1}, \ldots, I_{K-1}^{K-1} \right\}$.

We can now collect data sets from music sequences based on the tuple sets. Take Bach Chorales bmw26.6 as an example, Figure 5 shows part of the original music piece from Bach Chorales bmw26.6. If we set $m = 4$, and $n = 2$, we can get tuples as shown in Figure 6, and we will get tuples as in Figure 7 if we set $m = 5$, and $n = 3$.

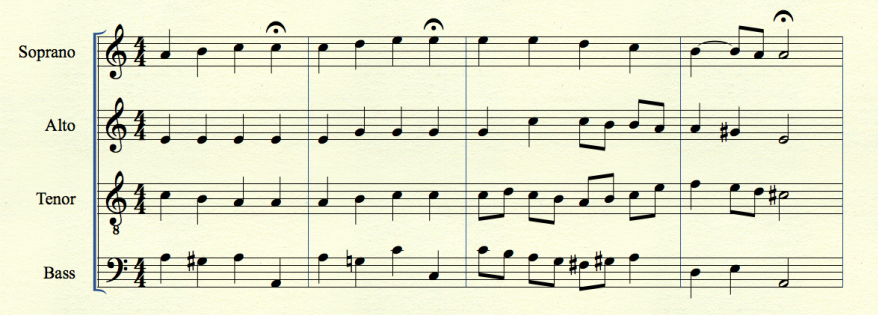


Fig. 5: *The first line of Bach Chorales bwm26.6*

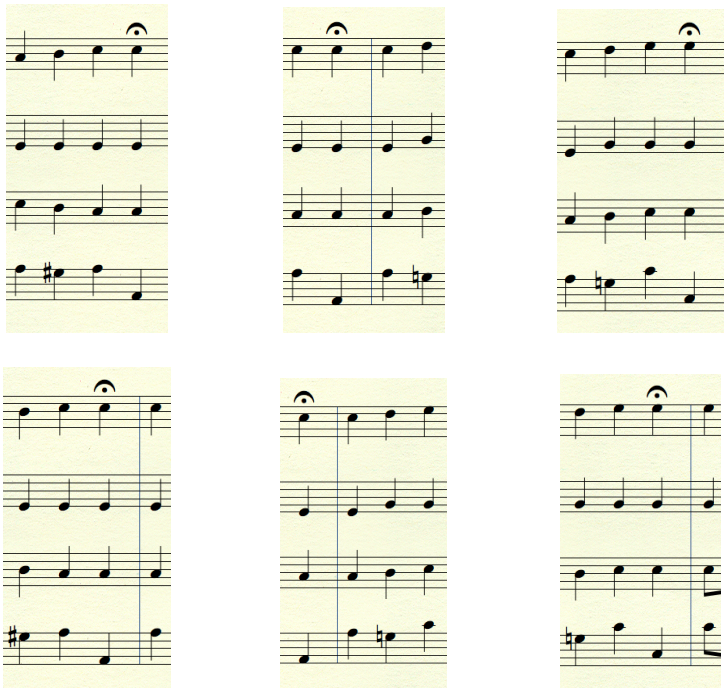


Fig. 6: *Sample tuples from Bach Chorales bmw26.6 when m = 4, and n = 2. The top three sample tuples are generated when i = 0, the bottom three tuples are generated when i = 1.*

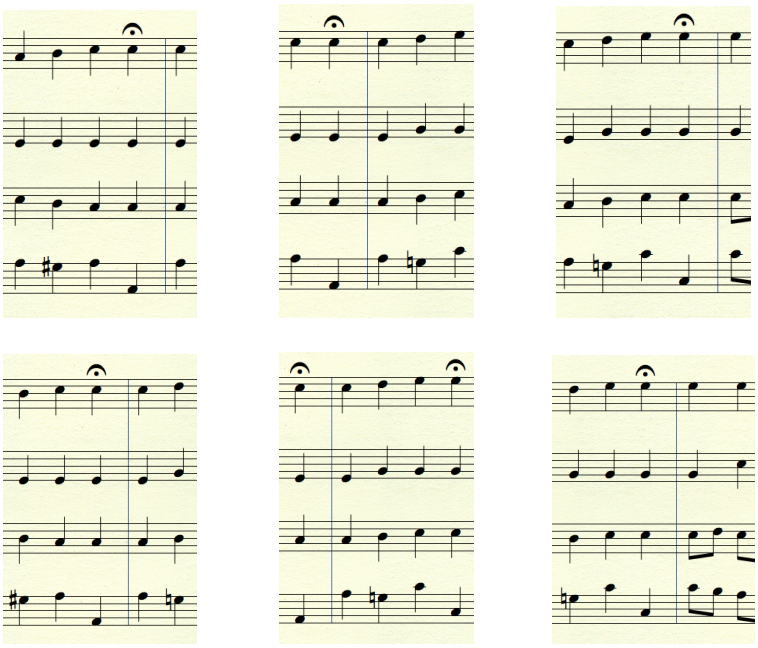


Fig. 7: *Sample tuples from Bach Chorales bmw26.6 when m = 5, and n = 3. The top three sample tuples are generated when i = 0, the bottom three tuples are generated when i = 1.*

Definition 15. *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*

$$\bigcup_{I\in\mathcal{I}} \pi_I(h)$$

If H is set of harmonic sequences, the indexed set $(\mathcal{I}, R)$ is then

$$(\mathcal{I}, R) = \bigcup_{h\in H}\bigcup_{I\in\mathcal{I}} \pi_I(h).$$

The following Figure 8 shows two short sequences, each one contains 8 chords. Figure 9 shows the results of joining the 4-tuples extracted from the two short sequences from Figure 8.

Now we can define the relation join for harmonic sequences.

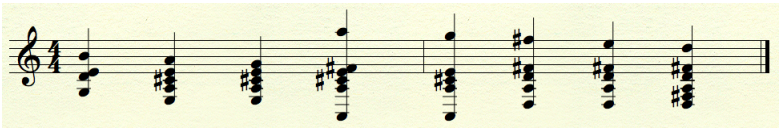
Definition 16. *If R is a set of m-tuples produced from all projections with index set $\mathcal{I} = \mathcal{I}(m,n)$, and if $I \in \mathcal{I}$ is an index set, (I,R) becomes an indexed relation with tuples of length m. Let $J = \cup_{I\in\mathcal{I}} I$, we then can get a set of new harmonic sequences by computing*

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

where $\otimes_{I\in\mathcal{I}}(I,R)$ is a set of $|J|-$ tuples.

<{'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'} >

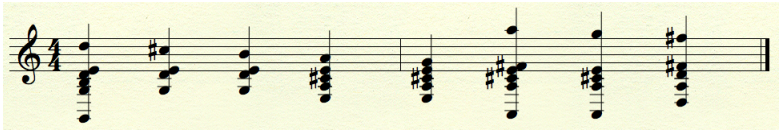
(a) *First short sequence written in tuple form*



(b) *First short sequence written in musical notation*

<{'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'} >

(c) *Second short sequence written in tuple form*



(d) *Second short sequence written in music notation*

Fig. 8: Shows two short sequences from Bach chorales.

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

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

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.

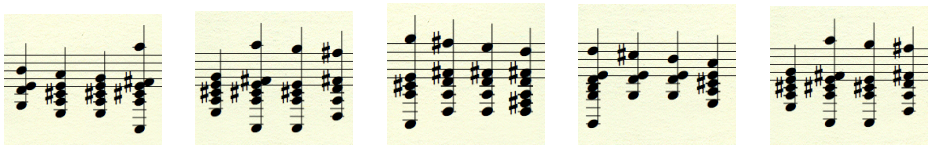
Now we could do **key constraint relation join**.

Definition 18. *Let* C_k *be a set of chords who are in the key of k, and* C^m *be the set of m-tuples of chords.* $R^b_k \subseteq R$ *contains all m-tuples of chords in which the first chord is in the key of k,* $'b'$ *means begin. Similarly,* $R^e_k \subseteq R$ *contains all m-tuples of chords in which the last chord is in the key of k,* $'e'$ *means end. That is,*

$$R^b_k = \{(c_1, c_2, \ldots, c_m) \in C^m \mid c_1 \in C_k\}$$

<{'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'}>

(a) *Shows the five 4-tuple generated from the two short sequences when* $m = 4$, $n = 2$ *and* $i = 0$ *in tuple form*



(b) *Tuple 1* (c) *Tuple 2* (d) *Tuple 3* (e) *Tuple 4* (f) *Tuple 5*

Fig. 9: *Shows the 5 4-tuples generated from the two short sequences in Figure 8 when* $m = 4$, $n = 2$, *and* $i = 0$. *Given* $m = 4$, $n = 2$, *and* $i = 0$, *sequence (b) of Figure 8 has 3 4-tuples, sequence (d) of Figure 8 has 3 4-tuples. The indexed relation formed by the corpus of sequences (b) and (d) could have 6 4-tuples, but in this case, two of them are identical. So there are only 5 4-tuples in the indexed relation produced by the two short sequences of Figure 8.*

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

Then key constraint relation join means computing

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

(J, S) *is the results of 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 19. *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*

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

Previous music generation methods try to generate music sequences $(x_1, x_2, \ldots, x_N)$ such that the probability that $(x_1, x_2, \ldots, x_N)$ is generated is equal to $P(x_1, x_2, \ldots, x_N)$. That is,

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

They use machine learning techniques to estimate a product decomposition of the joint probability. The joint product produced by a Markov Chain is $P(x_1, x_2, \ldots, x_N) = P(x_1) \prod_{n=2}^{N} P(x_n|x_{n-1})$. This is a special case of the more general form:

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

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 $f_k(x_i : i \in A_k)$.

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

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

Thus, in our method, $f_k(x_i : i \in A_k) > 0$ is ensured for any k, but we do not need to estimate the function f_k. Our 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 the patterns of music of different composers.

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 Chorales from the database of Music21. Every sequence 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 an harmonic chord sequence.

Secondly, we 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, we transform the chords in each sample into the integer indexes according to the dictionary.

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

Thirdly, we compute all tuples of chord from music pieces in the corpus. In this experiment, we set $\mathcal{I} = \mathcal{I}(4, 2)$, that is, $m = 4$, $n = 2$, $K = 14$, K is set to 14 to ensure the length of each output sample is 32, which is a reasonable length of a harmonic sequence. We only use the tuple sets when $i = 0$ in this experiment. Then we compute

$$\bigcup_{h \in H} \bigcup_{I \in \mathcal{I}} \pi_I(h)$$

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

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

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

Finally, we create mp3 files from the sequences generated.

The relation join procedure with $m = 4$ and $n = 2$, if done completely, generates over 24.12 million harmonic sequences in this experiment. We generate harmonic sequences using a random 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.

The following example shows how we generate a harmonic sequence of length 16 using tree search.

Example 6. The first three steps are the same as in the above experiments. The results after the three steps are all tuples of chords from the given music pieces. For the fourth step, instead of doing relation join as in experiment 1, we randomly pick a tuple from $\pi_{\{1,2,3,4\}}(I, R)$. The (I, R) we use in the following are extracted from Bach chorales. For example, we picked $(\{F\#4, D4, B3\}, \{G4, D4, B3\}, \{A4, D4, A3, F\#3\}, \{B4, D4, G3\})$. This tuple contains the first four chords of our synthetic sequence. Then we search over $\pi_{\{3,4,5,6\}}(I, R)$ to find a tuple that starts with the last two chords of the first tuple, that is chord $\{A4, D4, A3, F\#3\}$, and chord $\{B4, D4, G3\}$. If we find a tuple that matches like $(\{A4, D4, A3, F\#3\}, \{B4, D4, G3\}, \{A4, D4, F\#3, D3\}, \{A4, D4, G3, E3\})$, we join the two tuples we found into a 6-tuple $(\{F\#4, D4, B3\}, \{G4, D4, B3\}, \{A4, D4, A3, F\#3\}, \{B4, D4, G3\}, \{A4, D4, F\#3, D3\}, \{A4, D4, G3, E3\})$, then we search over $\pi_{\{5,6,7,8\}}(I, R)$ and find the next matched tuple, until we have 16 chords in the synthetic sequence. A sample synthetic sequence is as in Figure 10.

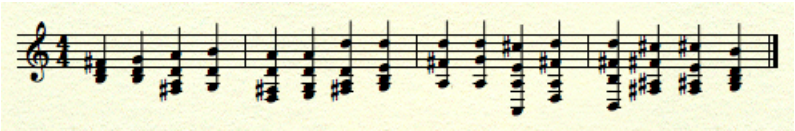


Fig. 10: *A sample synthetic music sequence generate using tree search, the duration is set to be quarter length as default.*

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 generate all the

results before sampling. After we have some samples, we can make them into $mp3$ files that can be listened to.

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 with overlap of 2 extracted in the third step in this experiment. Similarly to Experiment 1, we only use the tuple sets when $i = 0$ in this experiment.

In the first step, we extract harmonic sequence samples from given Bach chorales. A sample sequence is as following: $< \{'F4', 'C4', 'A3', 'F3', 0.5\}, \{'G4', 'C5', 'C4', 'G3', 'E3', 0.5\}, \{'C4', 'C5', 'G3', 'E3', 1.0\}, ... >$. 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.

The other four steps are almost the same as in experiment 1 except that we need to match the rhythm of the chords when doing relation join, while in experiment 1, we only need to match the chords. The relation join generates more than 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 6.

Table 6: *The number of sequences generated with key and scale constraint while* $m = 4$ *and* $n = 2$

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 7.

Table 7: *Shows the number of sequences generated with key and scale constraint while* $m = 5$ *and* $n = 3$.

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

Some samples from these experiments are also posted to the website: `http://haralick.org/music/music.html`. We also include some sample synthetic pieces in the Appendix B that could be played with.

5 Conclusion and Future Work

In this paper, we introduced a new music generation method based on relation join. We first break the input music into small segments, and then recombine them to generate new music sequences using relation join. Therefore, we do not have to estimate any probabilities or parameters from input music pieces. The idea behind our method is that the patterns of music sequences are carried in the progressions of them. Compared to other music generation methods, our method does not require any knowledge about music styles. Readers who are interested in hearing some of the generated sequences in midi format can find them on http://haralick.org/music/music.html.

In the future, we will try to generate music sequences of different composers and genres. We can also try to generate music sequences of mix-genres or mix-composers. For example, we can generate music sequences using the segments from Bach chorales and Beethoven's sonatas. With the information distance and similarity introduced, we are even able to generate music sequences with a certain level of similarities. For example, we can generate music that is very much like Bach chorales, which means the similarity is close to the original Bach chorales. It will also be interesting to see the most unlikely sequences that can be generated in a Bach style.

In order to do something like this, we will need to compute auto-similarity function for any harmonic sequences. In the following subsections, we describe how to do this voice by voice. One of the dimensions of similarity between harmonic sequences will be the cross similarity function, which we can define as the similarity between their respective auto-similarity functions.

5.1 Entropy

Under information theory context, entropy measures the information embedded within a random variable. Consider a categorical random variable X with probability distribution $p(x)$, the entropy can be calculated using the formula:

$$H(X) = -\sum_{x \in X} p(x) \log_b p(x)$$

where $\mathcal{X}$ is the set of possible values of X, and b is the base of logarithm. This formula is proposed by Shannon (1984) [13]. As a measure of uncertainty of a given event, it can also be extended to continuous random variable [14].

The joint entropy H(X, Y) of a pair of discrete random variables with a joint distribution p(x, y) is defined as:

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_b p(x,y)$$

where $\mathcal{Y}$ is the set of possible values for random variable Y.

The conditional entropy $H(Y|X)$ quantifies the information required for describing Y given X, that is, the average entropy of Y conditional on the value of X, averaged over all possible values of X. Let $H(Y|X = x)$ be the entropy when $X = x$,

$$\begin{aligned} H(Y|X) = -\sum_{x\in\mathcal{X}} p(x)H(Y|X=x) &= -\sum_{x\in\mathcal{X}} p(x)\sum_{yin\mathcal{Y}} p(y|x)\log_b p(y|x) \\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_b \frac{p(x,y)}{p(x)}. \end{aligned}$$

.

5.2 Mutual Information

In information theory, mutual information (MI) is a measure of mutual dependence of two given random variables, that is, MI quantifies the information revealed by one random variable through the other variable. If the two random variables are independent, MI is equal to 0. MI is closely related to the entropy of random variables. The MI of random variables X and Y is can be viewed as the information reduced if one random variable is given, that is,

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X).$$

The formal definition for MI is as following:

$$I(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

5.3 Information Based Distance and Similarity

With the definition of mutual information and entropy, we can define the distance between two random variables. A distance function $d : X \times Y \to \mathcal{R}^+$ is a metric if it satisfies the metric the following three conditions [19,20,15]:

- (A) Identity. $d(X, Y) = 0$ if and only if $X = Y$
- (B) Symmetry. $d(X, Y) = d(Y, X)$
- (C) Triangle inequality. $d(X, Z) + d(Z, Y) \geq d(X, Y)$

Several information based distance measures have been proposed [15,16,17,18].

$$d_1(X,Y) = H(X,Y) - I(X;Y) \tag{1}$$

$$d_1'(X,Y) = 1 - \frac{I(X;Y)}{max\{H(X),H(Y)\}} \tag{2}$$

$$d_2(X,Y) = H(X,Y) - min\{H(X),H(Y)\} \tag{3}$$

$$d_3(X,Y) = \frac{H(X,Y) - min\{H(X),H(Y)\}}{max\{H(X),H(Y)\}} \tag{4}$$

It can be proved that these each of four distance metric satisfy the three conditions. The proofs are in Appendix A.

The similarity $s(X,Y)$ can then be defined as:

$$s(X,Y) = 1 - d(X,Y) \tag{5}$$

5.4 Difference between Original Music and Synthetic Music

A note under music context is a small bit of sound with a dominant fundamental to introduce the frequency and harmonics sound, and can also be used to represent pitch class. A set of notes without scale is as following:

$$\mathcal{N} = \{C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#, B\}$$

From Section 3, we know that a music piece can then be represented by a tuple: $M = <N_1, N_2, \ldots, N_\tau, \ldots, N_T>, N_\tau \subseteq \mathcal{N}, i = 1, 2, \ldots, T$. Each element in the tuple is a chord, which is a subset of $\mathcal{N}$. Each chord has voices. We denote by N_t^v voice v of chord N_t.

The random variables in music are possible notes in the music piece. The joint probability can be written as:

$$p_\tau^v(x,y) = \frac{\#\{t | x \in N_t^v, y \in N_{t+\tau}^v\}}{T - \tau},$$

where $t \in \{1, \ldots, T\}$. The marginal of each random variable:

$$p_\tau^v(x) = \frac{\#\{t | x \in N_t^v\}}{T - \tau}$$

The mutual information can be calculated as:

$$I_\tau(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_\tau^v(x,y) \log \frac{p_\tau^v(x,y)}{p_\tau^v(x) p_\tau^v(y)}$$

Then we can calculate the distance using Equation 1, Equation 2, Equation 3, or Equation 4. For each τ, we can calculate similarity using Equation 5. So we can calculate a sequence of similarities from each piece. We call the sequence of similarities an auto similarity function.

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)
13. Shannon, C. E.: A mathematical theory of communication. Bell System technical Journal 27: 379-423 and 623-656. Mathematical Reviews (MathSciNet): MR10, 133e (1948)
14. Cahill, Nathan D.: Normalized measures of mutual information with general definitions of entropy for multimodal image registration. International Workshop on Biomedical Image Registration. Springer Berlin Heidelberg (2010)
15. Meila, Marina: Comparing clusterings an information based distance. Journal of multivariate analysis 98.5: 873-895 (2007)
16. Vinh, Nguyen Xuan, Julien Epps, and James Bailey. "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance." Journal of Machine Learning Research 11.Oct: 2837-2854 (2010)
17. Wikipedia.org/wiki/Mutual_information
18. Horibe, Yasuichi. "Entropy and correlation." IEEE transactions on systems, man, and cybernetics 5: 641-642 (1985)
19. Vitanyi, Paul MB, et al: Normalized information distance. Information theory and statistical learning. Springer US, 45-82 (2009).
20. Xu, Zeshui: Distance, similarity, correlation, entropy measures and clustering algorithms for hesitant fuzzy information. Hesitant Fuzzy Sets Theory. Springer International Publishing, 165-279 (2014).
21. Mackworth, Alan K. "Constraint satisfaction problems." Encyclopedia of AI, 285:293(1992).

A
Proofs of Distance Metrics

The proofs show that the four distance metrics we use in this paper satisfies the conditions:

- (A) Identity. $d(X, Y) = 0$ if and only if $X = Y$
- (B) Symmetry. $d(X, Y) = d(Y, X)$
- (C) Triangle inequality. $d(X, Z) + d(Z, Y) \geq d(X, Y)$

Proof. (A) and (B) are straightforward for all the four metrics give the properties of the entropy and mutual information . To prove the triangle inequality (C), we can first assume $H(X) \geq H(Y)$. Since $d(X, Y)$ is symmetric, the reverse case can be proved easily if we can prove one of the two cases ($H(X) \geq H(Y)$ or $H(Y) \geq H(X)$).

So we have only three cases left to consider: $H(X) \geq H(Y) \geq H(Z)$, $H(X) \geq H(Z) \geq H(Y)$, and $H(Z) \geq H(X) \geq H(Y)$.

- Case $H(X) \geq H(Y) \geq H(Z)$:
  - For $d_1(X, Y)$, first, we know that

$$\begin{aligned} d(X, Y)_1 &= H(X, Y) - I(X; Y) \\ &= H(X) + H(Y) - 2I(X; Y) \\ &= H(X|Y) + H(Y|X). \end{aligned}$$

    then

$$H(X|Z) + H(Z|Y) \geq H(X|Y, Z) + H(Z|Y) = H(X, Z|Y) \geq H(X|Y),$$

    so

$$\begin{aligned} d_1(X, Z) + d_1(Z, Y) &= H(X|Z) + H(Z|X) + H(Y|Z) + H(Z|Y) \\ &\geq H(X|Y) + H(Z|X) + H(Y|Z) \\ &\geq H(X|Y) + H(Y|X) = d_1(X, Y). \end{aligned}$$

  - For $d'_1(X, Y) = 1 - \frac{I(X;Y)}{max\{H(X),H(Y)\}}$,

$$\begin{aligned} d'_1(X, Y) &= 1 - \frac{I(X; Y)}{max\{H(X), H(Y)\}} \\ &= 1 + \frac{H(X, Y) - H(X) - H(Y)}{max\{H(X), H(Y)\}}. \\ &= 1 + \frac{H(X, Y) - H(X) - H(Y)}{H(X)}. \\ &= \frac{H(X, Y) - H(Y)}{H(X)} = \frac{H(X|Y)}{H(X)}. \end{aligned}$$

$$
\begin{aligned}
d'_1(X,Z) + d'_1(Z,Y) &= 1 + \frac{H(X,Z) - H(X) - H(Z)}{max\{H(X), H(Z)\}} + 1 + \frac{H(Z,Y) - H(Z) - H(Y)}{max\{H(Z), H(Y)\}} \\
&= 1 + \frac{H(X,Z) - H(X) - H(Z)}{H(X)} + 1 + \frac{H(Z,Y) - H(Z) - H(Y)}{H(Y)} \\
&\geq 1 + \frac{H(X,Z) - H(X) - H(Z)}{H(X)} + 1 + \frac{H(Z,Y) - H(Z) - H(Y)}{H(X)} \\
&\geq 1 + \frac{H(X|Z) - H(X)}{H(X)} + 1 + \frac{H(Z|Y) - H(Z)}{H(X)} \\
&\geq 1 + 1 + \frac{H(X|Y) - H(X) - H(Z)}{H(X)} \\
&\geq 1 + 1 + \frac{H(X|Y) - H(X) - H(X)}{H(X)} \\
&\geq 1 + \frac{H(X|Y) - H(X)}{H(X)} = \frac{H(X|Y)}{H(X)} = d'_1(X,Y).
\end{aligned}
$$

- For $d_2(X,Y)$, we first have $I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$, which means $H(X|Y) - H(Y|X) = H(X) - H(Y) > 0$, so

$$
\begin{aligned}
d_2(X,Z) + d_2(Z,Y) &= H(X,Z) - min\{H(X), H(Z)\} + H(Z,Y) - min\{H(Z), H(Y)\} \\
&= H(X,Z) - H(Z) + H(Z,Y) - H(Z) \\
&= H(X|Z) + H(Y|Z) \\
&\geq H(X|Z) + H(Z|Y) \\
&\geq H(X|Y) \\
&= H(X,Y) - H(Y) \\
&= H(X,Y) - min\{H(X), H(Y) = d_2(X,Y).
\end{aligned}
$$

- For $d_3(X,Y)$,

$$
\begin{aligned}
d_3(X,Z) + d_3(Z,Y) &= \frac{H(X,Z) - min\{H(X), H(Z)\}}{max\{H(X), H(Z)\}} + \frac{H(Z,Y) - min\{H(Z), H(Y)\}}{max\{H(Z), H(Y)\}} \\
&= \frac{H(X,Z) - H(Z)}{H(X)} + \frac{H(Z,Y) - H(Z)}{H(Y)} \\
&= \frac{H(X|Z)}{H(X)} + \frac{H(Y|Z)}{H(Y)} \\
&\geq \frac{H(X|Z)}{H(X)} + \frac{H(Z|Y)}{H(Y)} \\
&\geq \frac{H(X|Z)}{H(X)} + \frac{H(Z|Y)}{H(X)} \\
&\geq \frac{H(X|Y)}{H(X)} = d_3(X,Y).
\end{aligned}
$$

– Case $H(X) \geq H(Z) \geq H(Y)$: Similarly to the case $H(X) \geq H(Y) \geq H(Z)$.
– Case $H(Z) \geq H(X) \geq H(Y)$: Similarly to the case $H(X) \geq H(Y) \geq H(Z)$.

□

B

Harmonic Sequences in Scale C with 80 in quarterlength time duration

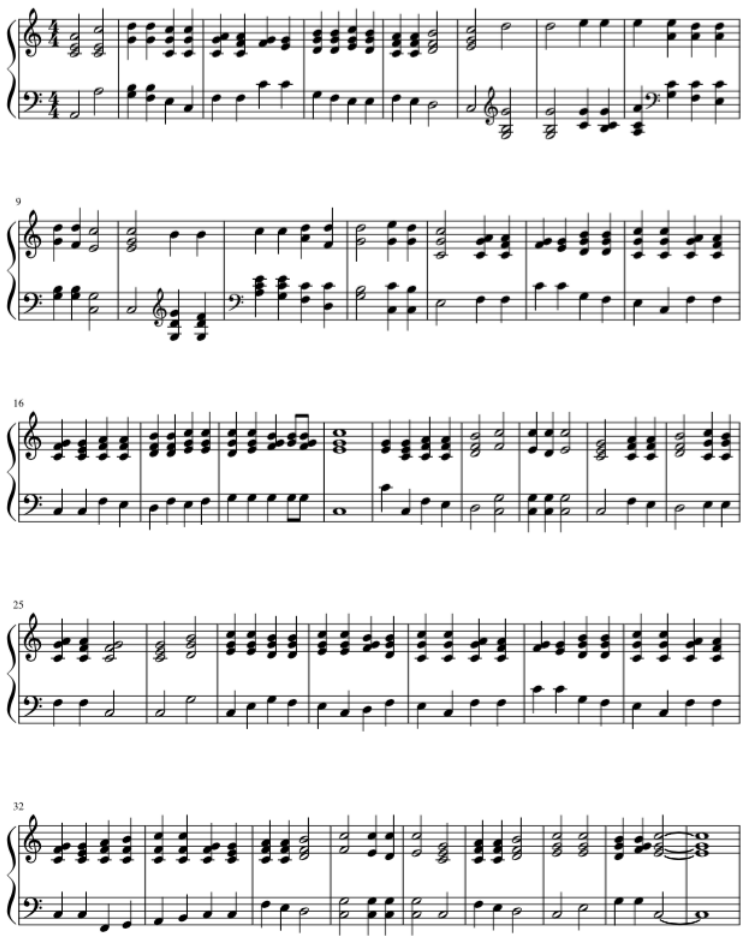


Fig. 11: *Synthetic harmonic sequence based on Bach Chorales with the scales fixed at C*

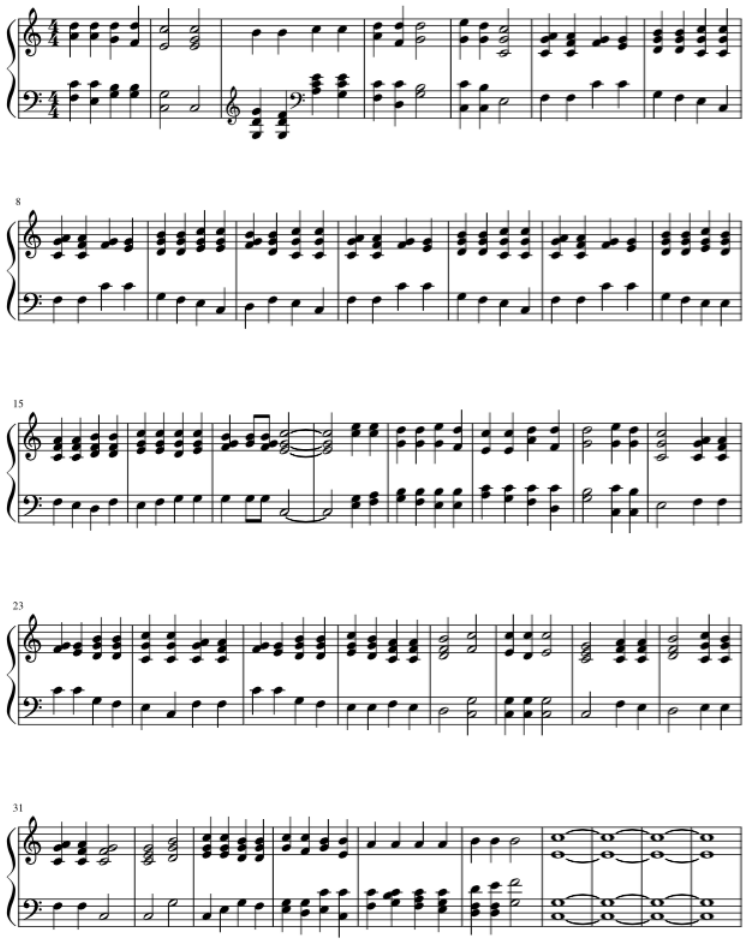


Fig. 12: *Synthetic harmonic sequence based on Bach Chorales with the scales fixed at C*