PART-INVARIANT MODEL FOR MUSIC GENERATION AND HARMONIZATION

Yujia Yan$^\flat$, Ethan Lustig$^\natural$, Joseph VanderStel$^\natural$, Zhiyao Duan$^\flat$
Electrical and Computer Engineering$^\flat$ and Eastman School of Music$^\natural$, University of Rochester
{yujia.yan, j.vanderstel, zhiyao.duan}@rochester.edu
ethan.s.lustig@gmail.com

ABSTRACT

Automatic music generation has been gaining more attention in recent years. Existing approaches, however, are mostly ad hoc to specific rhythmic structures or instrumentation layouts, and lack music-theoretic rigor in their evaluations. In this paper, we present a neural language (music) model that tries to model symbolic multi-part music. Our model is part-invariant, i.e., it can process/generate any part (voice) of a music score consisting of an arbitrary number of parts, using a single trained model. For better incorporating structural information of pitch spaces, we use a structured embedding matrix to encode multiple aspects of a pitch into a vector representation. The generation is performed by Gibbs Sampling. Meanwhile, our model directly generates note spellings to make outputs human-readable. We performed objective (grading) and subjective (listening) evaluations by recruiting music theorists to compare the outputs of our algorithm with those of music students on the task of bassline harmonization (a traditional pedagogical task). Our experiment shows that errors of our algorithm and students are differently distributed, and the range of ratings for generated pieces overlaps with students' to varying extents for our three provided basslines. This experiment suggests some future research directions.

1. INTRODUCTION

In recent years, there has been a growing interest in automatic music composition. Automatic music composition is a challenging problem, and it remains an open research topic regardless of many overblown statements in the press since the early days of artificial intelligence.

Apart from purely rule-based models that are difficult to craft, log-linear models, e.g., Hidden Markov Models (HMM), Conditional Random Fields (CRF), and Probabilistic Context-Free Grammars (PCFG) form a set of traditional methods for sequence modeling involving discrete variables (e.g., [13] [19] [18] [20]). When used for modeling music, they typically model each aspect of music (e.g., melody, harmony, durations) separately, or condition one variable on a small set of other variables (e.g, [1]). This is because, in music, when multiple aspects join together, the number of resulting combinations is prohibitively large, and the dataset is too small for learning every combination. Moreover, over-sized probability tables make inference extremely slow. Neural network based approaches solve this problem by expressing functions with a general high capacity approximator at the cost of higher computational requirements (relative to the small factorized model, but not always), less interpretability and fewer theoretic guarantees.

In [9] and [14], multi-layered LSTMs are used to model Bach's four-part chorales. For generation, the former uses Gibbs sampling and the later uses greedy search. In [10], a neural autoregressive distribution estimator is used to model the same Bach Chorales dataset, and for generation, authors compare Gibbs sampling, block Gibbs sampling, and ancestral sampling. In [22] and [6], Generative Adversarial Networks (GAN) are used to model and generate music pieces in their MIDI piano-roll form, and for generation, GAN based models sample the result directly without the need for an iterative sampling procedure.

However, most existing models, during training, adapt to specific music structures of the corpus being modeled. As our first attempt to extend the expressiveness of a music language model, we wonder if there is some invariance that can be exploited to obtain better generality. It is commonly believed that Bach wrote his chorale harmonizations by firstly writing out basslines for given melodies and then filling in inner voices (Alto, Tenor) [15]. Also, rules for each part (voice) share much in common, for example, a single part tends to move in the reverse direction after a leap. This motivated our idea of treating parts as the basic entity to model.

In this paper, we propose a part-invariant model for multi-part music [1]. Our generation framework follows the *Markov Blanket* formalism used in DeepBach [9]. Our model is a part-based model. As a basic consideration of counterpoint, each part should be in a good shape by itself, and when multiple parts are put together, the resulting

[1] Supplementary materials and some generation examples can be found at http://www.ece.rochester.edu/projects/air/projects/model0.html

aggregated sonority should be good. By *part-invariance*, we mean that the structure of our model explicitly captures the relationship among notes within every single part, and we share this structure with all parts of the score. A separate structure aggregates the information of how different parts would look like when joined together. As a result, our model is capable of expressing/processing music scores with any number of parts using a single trained model.

2. MULTI-PART MUSIC

In this work, we focus on music containing multiple monophonic parts (voices). For example, most of Bach chorales were written in the SATB format (Soprano, Alto, Tenor, and Bass), with each part containing a monophonic stream of notes. It is a traditional pedagogical practice to teach fundamental concepts of music theory by having students analyze and compose (i.e. "part write") this kind of music. When analyzing or composing music, we often separate a musical score into streams of notes [2], consciously or unconsciously. This part-separated form of music scores is easier to analyze and manipulate algorithmically, and many symbolic music analysis tasks use this separation as one of their preprocessing steps [7]. There are some existing approaches to perform part (voice) segmentation; see [7, 8] for more details. Therefore, our proposed technique focuses on encoding a part-segmented representation assuming the segmentation is known.

2.1 Representation

In traditional western music notation, durations of notes are derived by uniformly dividing a duration of a unit length recursively. Notes start and end on a subdivided position. It is thus reasonable to represent a music score as events on a grid, with each grid point representing a time frame. This process is commonly known as *quantization*. This practice can be seen in many works, e.g., [1, 9, 14, 22]. In this work, we keep the quantization step size fixed throughout the piece.

We encode two aspects of a music score: *pitch* and *metrical structure*. We make the following requirements for this representation:

1. This representation is able to encode a minimal set of musical notational elements, from which the reconstructed music score is human-readable.
2. Values at the same beat position under different quantization step sizes are the same.

Existing works make use of MIDI pitch numbers for encoding pitch. However, MIDI pitch numbers discard one element that is important for context determination: note spelling. In the proposed representation, pitch is represented by a tuple $(diatonic\ note\ number, accidental)$, where *diatonic note number* is the index of a note name with accidental removed (imagine the indices for white keys on a piano keyboard), and *accidental* has a range of $[-2, 2]$, that is, up to 2 flats and 2 sharps. For representing a whole note event, similar to [9, 17], we use a special continuation symbol, which is -1 in the diatonic note number field. For positions of rest notes, we artificially set their diatonic note number to 0. Accidentals are undefined in these two cases, therefore zeros can be filled in.

We encode the metrical structure into three simultaneous sequences sharing the same time resolution as the pitch frames: 1) *Bar Line* is a binary sequence encoding measure boundaries. A value of 1 is assigned to the frame at the first beat of a measure, and 0 is assigned elsewhere. 2) *Beat Level* encodes a frame's beat (sub-)division level in the metric hierarchy within a measure. Frames at the highest beat division level are assigned a value of 0; frames at the next level are assigned -1, etc. 3) *Accent Level* encodes the relative strength of beat positions of frames within a measure, with 0 representing the highest strength and -1 representing the second highest strength, etc. For example, for a classical 4/4 time signature, the frame at the first beat of a measure is assigned 0, the frame at the third beat is assigned -1, etc.

The first two encoding sequences work together to make it possible to reconstruct bar lines and the time signature. The third sequence further encodes metrical accents that are indicative of different music styles, and regular/irregular metrical structures.

Bar Line	1	0	0	0	0	0	0	0
Beat Level	0	-1	0	-1	0	-1	0	-1
Accent Level	0	-3	-2	-3	-1	-3	-2	-3

3. THE PART-INVARIANT MODEL

3.1 Model Architecture

Following the general practice of language models, our model predicts one symbol at a position given its (musical) context, that is,

$$P(x_{t,k} | context_{t,k}),$$

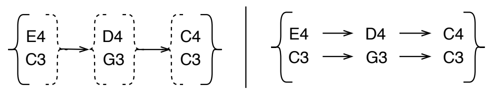
where t is the time frame index, k is the part index, and $x_{t,k}$ is the pitch representation at position (t, k). We further assume $context_{t,k}$ to be able to separate $x_{t,k}$ from influences of all other variables (*Markov Blanket* assumption). This *Markov Blanket* formalism is also used in [9].

For obtaining a vector summarizing the context for part k and frame t, after masking the symbol at the position (t, k) as a special *UNK* symbol, we first use a part-wise summarizer, which is a single-layered bidirectional RNN [2], to produce a *part-wise context vector* for each part. Then all part-wise context vectors are aggregated by one of reduction operations, e.g., max, min, sum, along the axis of part indexes, to produce an *aggregated context vector*. We also summarize the metrical structure (bar line, beat level, and accent level) with another single-layered bidirectional RNN to produce a *metrical context vector*. Finally, for time frame t, the part-wise context vector for part k, the

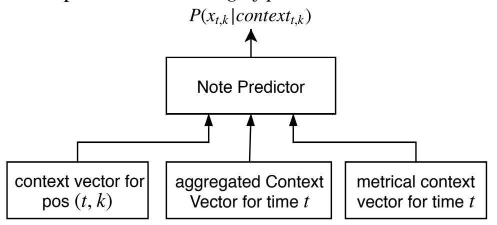
[2] *Bidirectional* here means that the output is a concatenation of outputs for the same time step from two RNNs with opposite directions.

aggregated context vector, and the metrical context vector are concatenated and fed into a feed-forward network with a $softmax$ output layer to obtain the final prediction $P(x_{t,k}|context_{t,k})$.

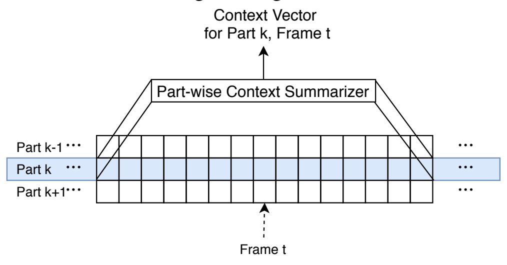
Our model is illustrated in Figure 1. Inputs to the part-wise context summarizer are vector embeddings described in Section 3.1.1, Inputs to the metrical context summarizer are raw metrical sequences, and the RNN structure used in our experiment is described in Section 3.1.2.



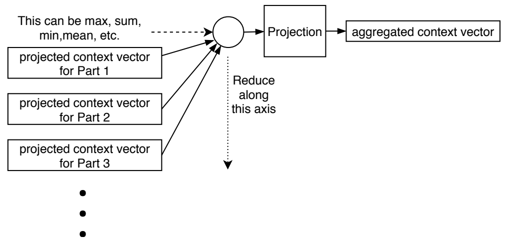
(a) *Sequence of Sets* vs. *Bag of Parts*. Our model is built upon the idea of *bag of parts*.


$P(x_{t,k}|context_{t,k})$
Note Predictor
context vector for pos (t, k)
aggregated Context Vector for time t
metrical context vector for time t

(b) Predicting a note given its context.


Context Vector for Part k, Frame t
Part-wise Context Summarizer
Part k-1 ···
Part k ···
Part k+1···
···
Frame t

(c) Part-wise context vector: each part is summarized by a bidirectional RNN.


This can be max, sum, min,mean, etc.
Projection
aggregated context vector
projected context vector for Part 1
projected context vector for Part 2
projected context vector for Part 3
Reduce along this axis

(d) Aggregated context vector: taking a reduction operation along the axis of projected part-wise context vectors and then projecting to a desired dimension.

Figure 1: Model Architecture.

3.1.1 Structured Pitch Vector Embedding

An embedding layer, which is usually the first layer of neural networks for modeling discrete symbols, learns a vector representation for each symbol. For embedding pitches, if each pitch is treated as a separate symbol, some general relationships that are already known (e.g., octave equivalence, intervals) will be lost. Therefore, we propose to use a factorized vector embedding representation (i.e., multiple terms in Eq. (1)) for each pitch for better generality.

For readers not familiar with embedding layers, one can treat V_k's below as lookup tables, each of which creates one entry (vector) for every possible value it takes.

The final vector embedding $V(p)$ is the sum of a series of embedding vectors, with each encoding a different "aspect" of a pitch.

$$\begin{aligned} V(p) = V_1(\text{diatonicPitchClass}(d)) + V_2(d) \\ +V_3(p) + V_4(\text{MIDI}(p)) \\ +V_5(\text{chromaticPitchClass}(\text{MIDI}(p))), \end{aligned} \tag{1}$$

where $p = (d, acc)$ is the pitch tuple defined in Section 2, with d being the diatonic note number and acc being the accidental; $\text{MIDI}(\cdot)$ is the MIDI pitch number; $\text{diatonicPitchClass}(\cdot)$ and $\text{chromaticPitchClass}(\cdot)$ wrap numbers according to octave equivalence; V is the final vector embedding; V_1, V_2, V_3, V_4, V_5 are vector embeddings for different aspects. These vector embeddings are jointly learned during training.

3.1.2 Stack Augmented Multiplicative Gated Recurrent Unit

The temporal dependency can be long for a representation using fine quantized time frames. In this work, instead of using standard LSTMs, we use a stack augmented multiplicative Gated Recurrent Unit as the RNN block. The GRU part implements the short-term memory. We choose the stack mechanism [11] for the long-term memory because of its resemblance to the pushdown automata, which has more expressive power than a finite state machine and is able to recognize context-free languages, which are often used to model some elements in music.

We make the following convention for our notation: unbolded lowercase letters, e.g, a, denote scalars; bolded lowercase letters, e.g, $\mathbf{x}$, $\mathbf{h}$, denote vectors; bolded uppercase letters, e.g, $\mathbf{W}$, $\mathbf{S}$, denote matrices.

The original GRU, as introduced in [4], transforms the input sequence of $\langle \mathbf{x}_t \rangle$ into a sequence of $\langle \mathbf{h}_t \rangle$, where t is the time step index:

$$\begin{aligned} \mathbf{r}_t &= \sigma(\mathbf{W}_r[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_r), \\ \mathbf{u}_t &= \sigma(\mathbf{W}_u[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_u), \\ \mathbf{c}_t &= \tanh(\mathbf{W}_c[\mathbf{x}_t; \mathbf{r}_t \odot \mathbf{h}_{t-1}] + \mathbf{b}_c), \\ \mathbf{h}_t &= \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \mathbf{c}_t, \end{aligned} \tag{2}$$

where $\mathbf{r}_t$ is the reset gate, $\mathbf{u}_t$ is the update gate, $\mathbf{c}_t$ is the update candidate, $\sigma(x) = \frac{1}{1+e^{-x}}$ is the Sigmoid function. $\mathbf{W}$'s and $\mathbf{b}$'s are all trainable parameters, representing weights and biases respectively, $\odot$ here represents element-wise multiplication, $[\mathbf{x}_t; \mathbf{h}_{t-1}]$ concatenates vectors into a longer column vector.

Multiplicative integration [21] adds quadratic terms into RNN update equations in order to improve the expressive power. In our implementation, we replace the equation for the update candidate with

$$\begin{aligned} \mathbf{c}_{t,x} &= \mathbf{W}_{cx}\mathbf{x}_t, \\ \mathbf{c}_{t,r1} &= \mathbf{W}_{cr1}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_{cr1}, \\ \mathbf{c}_{t,r2} &= \mathbf{W}_{cr2}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_{cr2}, \\ \mathbf{c}_t &= \tanh(\mathbf{c}_{t,x} \odot (\mathbf{c}_{t,r1} + 1) + \mathbf{c}_{t,r2}). \end{aligned} \tag{3}$$

A stack-based external memory for RNN is introduced in [11], which is reported to be able to learn some sequences that are not learnable by a traditional RNN, e.g, LSTM.

We denote the stack as a matrix $\mathbf{S}_t$, with dimensions of $N \times M$, where N is the length of one entry in the stack, and M is the capacity of the stack. In our implementation, a stack augmented memory performs the following procedure at each time step:

1. Fetch $\mathbf{v}_t$ from the stack by positional attention, which is a linear combination of columns in $\mathbf{S}_t$ with weights $\mathbf{k}_t$:

$$\begin{aligned}\mathbf{k}_t &= \mathrm{softmax}(\mathbf{W}_{\text{read}}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_{\text{read}}),\\ \mathbf{v}_t &= \mathbf{S}_t \mathbf{k}_t;\end{aligned} \tag{4}$$

where $\mathrm{softmax}(\mathbf{x}) = \frac{\exp(\mathbf{x})}{\mathbf{1}^T \exp(\mathbf{x})}$;

2. Augment the input with the fetched value:

$$\tilde{\mathbf{x}}_\mathbf{t} = [\mathbf{x}_t; \mathbf{v}_t], \tag{5}$$

and then run one step RNN with input $\tilde{\mathbf{x}}_\mathbf{t}$, which produces h_t;

3. Generate the input to the stack:

$$\mathbf{z}_t = \tanh(\mathbf{W}_z[\tilde{\mathbf{x}}_\mathbf{t}; \mathbf{h}_t] + \mathbf{b}_z); \tag{6}$$

4. Make decisions on how to update the stack:

$$\begin{bmatrix} a_{t,\text{no-op}} \\ a_{t,\text{push}} \\ a_{t,\text{pop}} \end{bmatrix} = \mathrm{softmax}(\mathbf{W}_\text{a}[\tilde{\mathbf{x}}_t; \mathbf{h}_t] + \mathbf{b}_a), \tag{7}$$

where a's are probabilities that sum up to 1 representing the probability of stack operations (no operation, push, pop);

5. Update the stack by expectation of operations:

$$\begin{aligned}\mathbf{S}_{t,\text{pushed}} &= [\mathbf{z}_t, \text{first}_k(\mathbf{S}_{t-1})],\\ \mathbf{S}_{t,\text{popped}} &= [\text{last}_k(\mathbf{S}_{t-1}), \mathbf{0}],\\ \mathbf{S}_t &= a_{t,\text{no-op}}\mathbf{S}_{t-1}\\ &+ a_{t,\text{push}}\mathbf{S}_{t,\text{pushed}}\\ &+ a_{t,\text{pop}}\mathbf{S}_{t,\text{popped}},\end{aligned} \tag{8}$$

where $\text{first}_k(\cdot)$ extracts the first k columns, and $\text{last}_k(\cdot)$ extracts the last k columns. Here $k = M-1$. Operator $[,]$ concatenates vectors/matrices horizontally.

3.1.3 *Context Aggregation: Obtaining Part-Invariance*

As mentioned above, the aggregated context vector is obtained by reduction operations on projected part-wise context vectors, and is then projected to the desired dimension.

$$\mathbf{C}_t^{\text{aggregated}} = \mathbf{W}_{\text{proj2}}(\bigoplus_{k=1}^{K} \mathbf{W}_{\text{proj1}}\mathbf{C}_{t,k}^{\text{part}}), \tag{9}$$

where $\mathbf{C}_t^{\text{aggregated}}$ and $\mathbf{C}_{t,k}^{\text{part}}$ are aggregated context vector and partwise context vector respectively, $\bigoplus_{k=1}^{K}$ denotes a reduction operator over k from 1 to K, where K is the number of parts, $\mathbf{W}_{proj1}$ and $\mathbf{W}_{proj2}$ are projection matrices for transforming the context vector into a higher dimension and back in order to improve the expressiveness of this reduction operation. In our experiment, we use max reduction. The proof of the universal approximation property for approximating a continuous set function when max reduction is used can be found in [3].

The reduction operation applied here produces a *continuous bag of parts* (*bag* means (multi-)set). This terminology draws similarity to *continuous bag of words* (CBOW, [16]), which averages all vector embeddings for all words (mean reduction) within a window to obtain the vector representation for this context. For comparison, existing works conceptually make use of *sequence-of-set* paradigm for context modeling (see Figure 1a), therefore the context model is confined to learning sequential relationships between sets. Our conceptual paradigm is on a different direction. We built a model for processing monophonic parts and a model for putting them into a bag. One important feature for doing this is that it allows learning shared properties of parts. Also, the ordering of parts, which is redundant for a context encoder, is discarded and only the content information of all parts is aggregated. As a result, it reduces the model complexity required.

3.2 Sampling and Generation

After training the Markov blanket model for approximating the probability of a note conditioned on its context, $P(x_{t,k}|context_{t,k})$, the process of generation is performed by Gibbs sampling with an annealing schedule. This procedure is almost the same as the one used in [9].

Firstly, we initialize notes $x_{t,k}$ of all positions in the empty parts randomly. Then we iterate:

1. Randomly or deterministically select the next position (t, k) that is not fixed [3] to sample;
2. Sample new $x_{t,k}$, according to

$$\tilde{P}(\,\cdot\,|context_{t,k}) \propto (\,P(\,\cdot\,|context_{t,k})\,)^{1/T}, \tag{10}$$

i.e, the annealed distribution with temperature $T > 0$;

For vanilla Gibbs sampling, $T \equiv 1$. However, as pointed out in [9], conditional distributions outputted are likely to be incompatible and there is no guarantee that the Gibbs sampler will converge to the desired joint distribution.

In Gibbs sampling with an annealing schedule, the temperature starts from a high value and gradually decreases to a low value. By incorporating this annealing scheme, the algorithm can escape from initial bad values much easier at the beginning, and the average likelihood for the selected new samples increases as the temperature decreases. For illustration, in the limiting case that $T \to 0$, the algorithm

[3] Fixed positions are used as conditions. For example, if the task of melody harmonization, the melody part is fixed.

greedily selects new samples that maximizes the local likelihood.

Since in our model parts are orderless, the generated result does not ensure all parts in their usual notated staffs. Also, the imperfection of Gibbs sampler often makes the configuration get stuck in a region where voice crossing occurs even if parts in the training set rarely cross. In the experiment, we enforce one constraint during sampling as a workaround: in each time frame, the pitch of each part cannot go above/below the part that is immediately above/below it, i.e., no voice crossing is allowed. This constraint is achieved by limiting the range of candidates to sample. How to design a better sampling procedure is left for future investigation.

4. EXPERIMENT

4.1 Training

4.1.1 Dataset

We trained our model on the Bach Chorale dataset included in Music21 [5]. We chose this dataset to perform our experiment for the following reasons: firstly, it is publicly available; secondly, it matches the objective evaluation methods we designed [4]; thirdly, there is no need to perform voice separation in this dataset. Different parts are separately encoded in the file format.

We performed data augmentation by transposition with a range such that the transposed piece is within the lowest pitch minus 5 semitones to a highest pitch plus 5 semitones for the whole dataset. Enharmonic spellings are resolved by selecting the one that creates the minimum number of accidentals for the entire transposed piece.

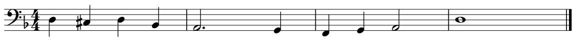
4.1.2 Model Specification

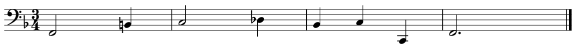
In our experiment, we use a quantization step size of a sixteenth note. Embedding layers have a dimension of 200. We use single-layered RNNs as the partwise context summarizer and metrical sequence summarizer. All RNNs have a hidden state size of 200, stack vector length 200, stack size 24. The intermediate dimension for the part aggregating layer is 1000. The final predictor is a feed-forward neural network with 3 layers, each of which contains 400 hidden units. The final softmax layer has a dimension of 400, each corresponding to a specific pitch tuple (diatonic note number, accidental). We use cross entropy as the loss function. Curriculum learning is used in our training: we started from a small half-window width of 8 and gradually doubled the half-window width to a maximum of 128. During training, pitches within the context window are randomly set to a rest with probability 0.1. All layers except for the RNN layers use a dropout rate of 0.1.

In our evaluation, we use an half-window width of 64 for generation. We use a simple linear cooling schedule to decrease the temperature from an initial value of 1.2 to 0.25. The total number of iterations is selected such that every position is sampled 40 times.

Music scores are reconstructed by directly using accidentals, diatonic note numbers and the original encoded metrical sequences. Key signatures and clefs are automatically determined by Music21's [5] built-in functions.

4.2 Evaluation


(a) bassline1


(b) bassline2

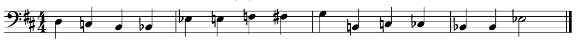

(c) bassline3

Figure 2: Basslines used in our evaluation.

To perform evaluation, we compared our algorithm's harmonizations of basslines with harmonizations of those same basslines completed by music students. We used three basslines which vary in difficulty, ranging from diatonic (bassline 1) to moderately chromatic (bassline 2) to highly chromatic (bassline 3). [5] For each bassline, our algorithm generated 30 outputs, for a total of 30*3 outputs. As a side note, 4 bars is the usual length for a harmonization exercise. This length is different from lengths of pieces in the training set.

We recruited 33 second-semester sophomore music majors, offering them extra credit for harmonizing each bassline. We gave each student a .xml file containing the three basslines, with three blank upper staves . We instructed students to harmonize each of the basslines in four-part, SATB chorale style, following the usual rules of voice-leading and harmony. We used valid responses from 27 students (those not empty and returned timely) in the following evaluation tasks.

We recruited two teams for evaluation: graders and listeners. The graders were three music theory PhD students. They were given the 57 valid outputs ($57 * 3$ in total) in .pdf format; we created a grading rubric [6]. A deduction less than 0 was computed by each grader for each output. The lower the value, the greater the number of errors. One graded example can be found in Figure 3.

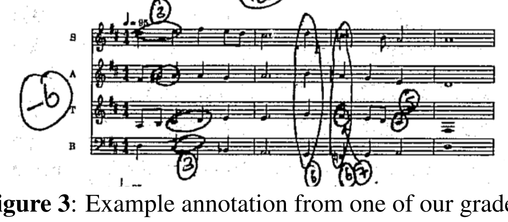


Figure 3: Example annotation from one of our graders.

[4] The objective evaluation follows rules used in textbook part writing, which are greatly influenced by Bach Chorales, however, these rules are not strictly followed by Bach himself.

[5] Basslines 1 and 2 were taken from Exercises 10.3C and 21.2, respectively, from [12]. In Bassline 2, the B was originally a Bb in [12], but we changed it to increase chromaticism. Bassline 3 was created by us, and intended to represent highly modulatory chromatic harmony.

[6] The rubric is typical of traditional music theory textbooks and classes. For the detailed rubric, see the supplementary website.

While the grading method was fairly objective (correlations between error values from the three graders were .85, .88, and .92), we also wanted subjective ratings. In addition, we recruited a listening team of another three music theory PhD students. We gave them the same 57 * 3 outputs in .mp3 format, synthesized with a software piano synthesizer and tempo 93, and these instructions:

For each output, answer the following four questions:

1. *As you listen, how much are you enjoying this solution (on a scale of 1 to 4, where 1 = not enjoying at all and 4 = greatly enjoying)?*
2. *As you listen, how confident are you that this solution is by a computer vs. a sophomore (on a scale of 1 to 4, where 1 = probably a computer and 4 = probably a sophomore)?*
3. *As you listen, to what extent does this solution conform to textbook/common-practice voice-leading and harmony (on a scale of 1 to 4, where 1 = not very idiomatic and 4 = quite idiomatic)?*
4. *Please share any other comments or thoughts (for example, why does it sound like it's a computer vs. a sophomore?)*

To summarize, for each output we had 3 gradings (1 value * 3 graders) and 9 subjective ratings (3 ratings*3 listeners) plus additional open-ended comments.

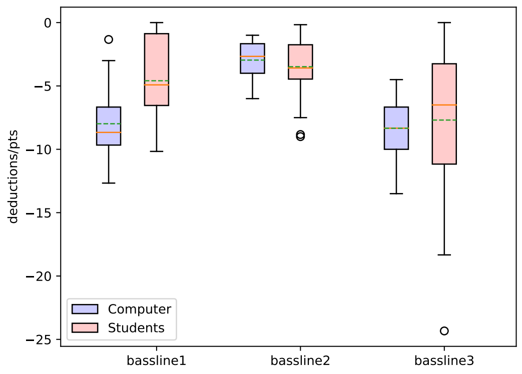
To minimize bias, graders only received .pdf outputs; listeners only received .mp3 outputs. The outputs were presented to the graders and listeners in random order. Both teams were blind to the output source (computer or student), and were allowed to take as much time as they needed to make their assessments.

Our experiment result is summarized in Figure 4. Our experiment shows that gradings and listening ratings for our algorithm and students overlap to different extents (our algorithm performs best on the second bassline). For the listening test, our algorithm consistently performs a bit worse than average second-year second-semester music majors.

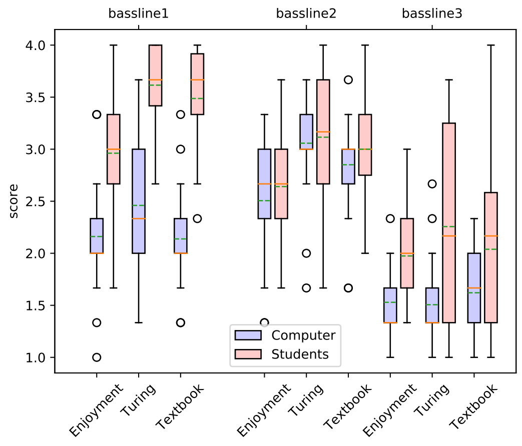
The comments from the listener who contributed most of the open-ended comments (question 4) suggest that the presence of *tonality* was one of the main factors in their Turing judgements. This listener attributed harmonizations that feature small stylistic errors (e.g., oddly repeated notes, parallel voice leading, etc.) to both human and computer, but those harmonizations that sounded resolutely tonal were only attributed to humans. Another listener seemed to ground their judgments on a different feature: “A lot of the ones I think are computer-generated do cadences super well.” Indeed, for the most part the computer did generate well-formed cadences.

By examining the detailed responses from our graders, we have the following rudimentary observations:

1. Errors of our algorithm and students are differently distributed.
2. Parallel octave/fifth (Error 3) is one frequent error produced by our algorithm, more often than students. This type of error is also observed in generation examples shown in [9].



(a) Results for the objective grading test.



(b) Results for the subjective listening test.

Figure 4: Objective and subjective comparisons between our algorithm's and music students' harmonization on the three basslines.

3. Our algorithm produces more non-stylistic progressions (Error 6 and Error 7). In our algorithm, it is observed that, smooth/melodic voice leading may sometimes suppress the requirement of the vertical sonority.
4. Students are much more likely to exceed the octave range limit between nearby upper voices (Error 9)

From our experiment result, it is revealed that the proposed algorithm cannot learn what is bad/incorrect just by watching correct examples. Therefore, there is a need to train with negative examples. Our experiment provides useful data for future development.

5. CONCLUSION

In this work, we proposed a part-invariant model for music generation and harmonization that operates on multi-part music scores, which are scores containing multiple monophonic parts. We trained our model on Bach Chorales dataset. We performed objective and subjective evaluations by comparing the outputs of our algorithm against the textbook-style part writings of undergraduate music majors. Our experiment result provides insights and data that will be useful for future development.

6. REFERENCES

[1] Moray Allan and Christopher Williams. Harmonising chorales by probabilistic inference. In *Advances in neural information processing systems*, pages 25–32, 2005.

[2] Albert S. Bregman. *Auditory scene analysis*. MIT Press, 1996.

[3] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[4] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[5] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data.

[6] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *AAAI-18 AAAI Conference on Artificial Intelligence*, 2018.

[7] Patrick Gray and Razvan C Bunescu. A neural greedy model for voice separation in symbolic music. In *International Society for Music Information Retrieval Conference (ISMIR 2016)*, pages 782–788, 2016.

[8] Nicolas Guiomard-Kagan, Mathieu Giraud, Richard Groult, and Florence Levé. Comparing voice and stream segmentation algorithms. In *International Society for Music Information Retrieval Conference (ISMIR 2015)*, pages 493–499, 2015.

[9] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *International Conference on Machine Learning*, pages 1362–1371, 2017.

[10] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron C. Courville, and Douglas Eck. Counterpoint by convolution. In *ISMIR*, pages 211–218, 2017.

[11] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pages 190–198, 2015.

[12] Steven G. Laitz. *Writing and analysis workbook to accompany The complete musician: an integrated approach to tonal theory, analysis, and listening, 3rd edition*. Oxford University Press, 2012.

[13] Victor Lavrenko and Jeremy Pickens. Polyphonic music modeling with random fields. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 120–129. ACM, 2003.

[14] Feynman T. Liang, Mark Gotham, Matthew Johnson, and Jamie Shotton. Automatic stylistic composition of bach chorales with deep lstm. In *ISMIR*, pages 449–456, 2017.

[15] Robert L Marshall. How JS Bach composed four-part chorales. *The Musical Quarterly*, 56(2):198–220, 1970.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[17] Franois Pachet, Alexandre Papadopoulos, and Pierre Roy. Sampling variations of sequences for structured music generation. In *ISMIR*, pages 167–173, 2017.

[18] Martin Rohrmeier. A generative grammar approach to diatonic harmonic structure. In *Proceedings of the 4th sound and music computing conference*, pages 97–100, 2007.

[19] Ian Simon, Dan Morris, and Sumit Basu. Mysong: automatic accompaniment generation for vocal melodies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 725–734. ACM, 2008.

[20] Andries Van Der Merwe and Walter Schulze. Music generation with markov models. *IEEE MultiMedia*, 18(3):78–85, 2011.

[21] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan R Salakhutdinov. On multiplicative integration with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 2856–2864, 2016.

[22] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In *Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR2017), Suzhou, China*, 2017.