GENERATING MUSICAL CONTINUATIONS WITH REPETITION

Sebastian VELEZ DE VILLA (sebastian.velezdevilla@epfl.ch) [1],
Andrew MCLEOD (andrew.mcleod@epfl.ch) (0000-0003-2700-2076) [1], and
Martin ROHRMEIER (martin.rohrmeier@epfl.ch) [1]

[1] *Digital and Cognitive Musicology Lab*, *Digital Humanities Institute*, **École Polytechnique Fédérale de Lausanne**, Switzerland

ABSTRACT

Repetitions play a central role in music, be they repeated themes, harmonies or rhythmic repetitions. They can differ in many ways: vertically, by shifting notes up or down in pitch, time-dilated, by slowing or accelerating them, or just slightly different, for example with ornamentation or removed notes. This work focuses on explicitly generating continuations with patterns given a musical excerpt. We compare two different ways of finding such repetitions, and the found patterns are used to help generate music that is more musically well-formed than a pattern-agnostic approach. Quantitative results show an improvement in performance for both of our algorithms over a pattern-agnostic baseline, with the more sophisticated algorithm exhibiting the most promising results. Qualitatively, it still lacks some creativity, as the model only creates patterns or notes that already exist in the given excerpt, which is particularly an issue for pieces that do not exhibit a large amount of repetition.

1. INTRODUCTION

This paper focuses on monophonic music generation, and in particular on generating music with repetition. Repetition is a fundamental component of musical form [1, 2], from classical music to modern-day pop. Cognitively, it is a major component of a listener's experience with a piece of music, where the prediction and recognition of repeated themes and motives—even (and in some cases in particular) for non-exact repetitions—can lead directly to the enjoyment of the listening experience [3, 4]. Therefore, the inclusion of repetition must form an essential component in any music generation system.

The evaluation of generated music is an extremely subjective and difficult task [5]. Therefore, we concentrate on the first subtask of the MIREX Patterns for Prediction (PP) task [1], whose goal is to develop algorithms that take a short excerpt (a prime) of a piece of music and produce a continuation: some notes that will follow the prime. This allows us to explore music generation with repetition, and has a well-defined quantitative evaluation metric (though we also perform a qualitative evaluation of our results).

The main challenge for our approach is to develop a robust way to find patterns with variation in music. By "robust", we mean an algorithm that can find repeated sequences that would be considered very similar by human listeners: it could be a repeated theme played at a different pitch or tempo, one with pitch or rhythmic variations, or one with added or removed notes. This task is not trivial, even for humans, as the perception of such patterns can be very subjective: even annotators do not have perfect agreement with each other [6].

For a computer, finding such repeated patterns efficiently is even more difficult. It is important to note that hierarchical models of musical form have indeed shown the ability to capture some longer-term, overarching properties of repetition in music, for example in the task of splitting a piece into sections [7]. However, this is quite a different task than detecting the specific local repetitions in which we are interested, and can rely on more global structures such as the harmonic texture of a piece.

A naive approach to finding local repetitions is to compare all subsets of the short excerpt with the whole set, and count how many times each one of them appears in the prime. This solution works for exact matches, but is slow, and doesn't work when the patterns are not exact. It is often the case that themes, or important parts or even sections of the music, are repeated, but those themes commonly undergo variation, transformation or thematic-motivic work throughout a piece. Thus, while the ability to find exact repeats in music is a *necessary* component for our system, it is not a *sufficient* way to ensure that all repeated patterns are found, so we use a more sophisticated approach. In other words, while finding perfect matches is feasible, such patterns are not sufficient for generating music.

Once these repeated patterns are found in our given prime, a continuation must be generated. In recent years, many deep learning approaches have been applied to the task of music generation (e.g., [8–11]). However, such models have a couple of drawbacks for our purposes. For one, we want our model to be adaptable in order to generate music that is strongly informed locally by the given prime. One potential solution for this would be to combine a long-term model (which is trained on a large corpus of music to learn general rules of tonality and form) with a short-term model (which learns the local structure

[1] https://www.music-ir.org/mirex/wiki/2019:Patterns_for_Prediction

Pitch class	C	D	E	F	G
C	0.25	0.5	0.25	0	0
D	0	0	1.0	0	0
E	0.5	0	0	0.5	0
F	0	0	0	0	1.0
G	0	0	1.0	0	0

Table 1: Transition matrix of Figure 1.

specific to a single musical piece). This approach been used by statistical models of music (e.g., [12, 13]); however, it is more difficult to apply this to large networks given the large amounts of training data they are trained on initially and their large parameter count in comparison to such a short prime. Furthermore, given the relatively short length of the continuations required by our chosen task, it is unclear whether such a long-term model would even help. Secondly, with such large black-box-type models, it is difficult to ensure an explicit generation of repeated patterns, as is feasible with simpler alternatives (although some work has been done in this direction, with the goal of enforcing local structural constraints on symbolic music generation based on a template piece [14]).

Therefore, in this work we instead concentrate on statistical approaches to music generation. Specifically, we take existing algorithms for detecting patterns with variations [15, 16], and investigate the results of applying them (with only minor changes) to the task of music generation using a single-order Markov model. Our approaches perform well against the task's simple baseline, described in section 2, but can still lack creativity in some cases, particularly when the prime is not very repetitive.

2. RELATED WORK

In [17], the authors discuss how a computer can generate music using statistics. The basic idea uses Markov chains, i.e., a succession of states, where each state only depends on the previous one. A simple music generation algorithm can be extended from that idea. For example, in [18], a Markov model is trained on a large corpus of pieces and then used to generate a continuation of an input excerpt (similar to our task). However, this model draws from pre-learned patterns from its training corpus, rather than patterns taken from the prime directly, as we wish to do.

Consider the simple melody in Figure 1 (the first 4 bars of "Frère Jacques") as an example. Take each pitch class as a state.

To calculate the transition probabilities for each state, a possible algorithm could be formulated in the following way: create a transition matrix, and for each pitch, count how many times a note of that pitch is followed by notes of other (or the same) pitches, and divide by the number of times that pitch appears (disregarding the last note which, by definition, doesn't have a next state). This generates the transition matrix shown in Table 1, which should be read line then column. For example, "D is followed by E with probability 1.0". If the excerpt in Figure 1 is taken as a prime, this table could then be used to generate the continuation by taking the last state (here, "G"), and successively drawing the next note from the distribution in the corresponding row in the table. (A possible output for a 4-note continuation would be "E C E F".) This simple generation algorithm can thus only generate states that appear in the prime sequence (i.e., the input), and can only generate transitions that already exist (e.g., we can never have a C followed by a G, and we can never have a B at all). There is therefore only a very limited notion of creativity or musical structure in this case, only a generation of repetitions of what already exists. To a human listener, most of the outputs from this algorithm sound relatively simple. However, it is fast, and can run in $\mathcal{O}(n+k)$, where n is the length of the sequence, and k is the number of generated states.

In the previous example, the Markov model was used to generate only pitches, but it can also be extended for durations, or pairs of pitches and durations by simply defining the state space differently. In this work, we compare our system against a baseline Markov model trained to generate sequences of (pitch, inter-onset-interval) pairs (i.e., the state space of the model consists of such tuples) [17]. This model is also used as a baseline for the Mirex PP task [2].

2.1 Pattern Detection

In order to improve the basic model of [17], the authors propose a solution based on pattern discovery and pattern inheritance, where a pattern is defined as a set of notes which is repeated, shifted, or time-dilated in the piece. For example, in Figure 1, one pattern would be "C D E C", since it is repeated twice.

The most basic method for pattern discovery is a naive string-based approach which finds only exact repetitions. Many more sophisticated methods exist, none of which is clearly the best in all cases, and they are often designed for very different goals. For example, recent overview of the highly related task of symbolic melodic similarity (in which systems measure the similarity between two musical excerpts), can be found in [19]. In [20], pattern detection was used to perform melodic segmentation by searching for patterns in pattern occurrences. Since they are designed with specific downstream tasks in mind, the above approaches tend to be more complex and less transparent than basic pattern detection algorithms. We therefore draw instead from the basic pattern detection literature, leaving more sophisticated approaches for finding different types of patterns for future work. More thorough discussions on some of these methods (in various contexts) can be found in, e.g., [6, 21–23].

We choose to use a pattern-detection algorithm based on SIA [15] [3], since it was simple to implement, adapt, and investigate for our purposes. We describe SIA and its integration into our generation process in the following section.

[2] `https://glitch.com/@tomthecollins/wi-mir-2020-workshop`

[3] Although SIA is well-adapted for polyphonic textures, we use it here on monophonic input since it works well, is not slow (given the lengths of our primes), and will allow us to more easily adapt to polyphonic music in future work.

Figure 1. The first 4 bars of “Frère Jacques”.

3. DESIGN AND IMPLEMENTATION

3.1 Input format

Our pattern-informed music generation algorithms take as input monophonic sequences of midi notes, which are tuples of pitch (represented as MIDI note number), onset, duration and velocity attributes. Before the detection of patterns and generation, each prime is transformed so that it does not include rests, making both more straightforward. The pitches and onsets remain unchanged, only the durations are updated, so that each note ends when the next note begins. When generating the next state, the starting time of each note is set to the offset time of the previous note.

3.2 Pattern detection

In order to find such patterns, we decided to follow two approaches.

1. A *string-based* approach, which finds exact patterns.
2. A *translation-based* approach, which finds exact patterns, as well as vertically shifted patterns.

Consider the sequence “1 2 3 4 2 3 4 5” consisting of eight states. The string-based solution would only find the pattern “2 3 4”, whereas the translation-based algorithm would find the pattern “1 2 3 4”, knowing that it is shifted up by one to obtain “2 3 4 5”.

3.2.1 String-based pattern recognition

In this case, the notes of the prime are transformed into a list of (pitch, duration) tuples, and the state-space of the Markov model consists of such tuples. The algorithm then uses a string-based pattern matching approach, and finds only exact patterns, defined as a sequence of these (pitch, duration) tuples which appears at least twice in the prime. A pattern detected by this algorithm for Figure 1 would be the first four notes.

3.2.2 Translation-based pattern recognition

As opposed to the string-based approach, this algorithm first transforms the notes of the prime into a list of (pitch, onset) tuples. Then, it uses *translation vectors* to find the maximum translatable pattern: that is, a sequence of notes that can be shifted either vertically (in pitch) or horizontally (in onset time) in the score to find a matching sequence. As opposed to the string-based approach, this method uses onset instead of duration, which is needed to calculate the translation vectors, described below. This algorithm is based on SIA [15], with the difference that we consider only contiguous patterns, which helps for the generation process. We instead leave non-contiguous pattern-based generation for future work.

To make things simple, we will detail this process step by step, again using the simple example of “Frère Jacques” (Figure 1). For each note, we first calculate its translation vector with respect to all following notes in the sequence. Specifically, let n_i and n_j be ith and jth notes of the prime, where $i < j$. Then the translation vector of n_i to n_j is calculated as in Equation 1, where the pitch is represented by its MIDI note number:

$$\begin{pmatrix} n_j.onset - n_i.onset \\ n_j.pitch - n_i.pitch \end{pmatrix} \tag{1}$$

Considering the first two bars of “Frère Jacques,” This results in the table shown in Table 2. (Durations are measured in whole notes here, but any other representation would be equivalent, as long as it is consistent throughout a prime.) Here, the table should be read starting with columns and then rows (e.g., the first note (0.0, 72) can be transformed into the second note (0.25, 74) by adding the translation vector (0.25, 2). Then for each translation vector that appears at least twice in the table (signified by colors), we create a sequence of those notes which have this vector in their column. Sorting the translation vectors by the length of the resulting sequence of notes results in:

- $\begin{pmatrix} 0.25 \\ 2 \end{pmatrix}$: $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$, $\begin{pmatrix} 0.25 \\ 74 \end{pmatrix}$, $\begin{pmatrix} 1.0 \\ 72 \end{pmatrix}$ and $\begin{pmatrix} 1.25 \\ 74 \end{pmatrix}$.
- $\begin{pmatrix} 1.0 \\ 0 \end{pmatrix}$: $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$, $\begin{pmatrix} 0.25 \\ 74 \end{pmatrix}$, $\begin{pmatrix} 0.5 \\ 76 \end{pmatrix}$ and $\begin{pmatrix} 0.75 \\ 72 \end{pmatrix}$.
- $\begin{pmatrix} 0.5 \\ 4 \end{pmatrix}$: $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$ and $\begin{pmatrix} 1.0 \\ 72 \end{pmatrix}$.
- ...
- $\begin{pmatrix} 1.25 \\ 2 \end{pmatrix}$: $\begin{pmatrix} 0.0 \\ 72 \end{pmatrix}$ and $\begin{pmatrix} 0.25 \\ 74 \end{pmatrix}$.

Each of these sequences is a potential repeated pattern. They are filtered to keep only those sequences which contain only contiguous notes, like the second one in the enumeration above (shown in purple in the table), and unlike the first one (in red). This filtering eliminates any “split” patterns whose beginning and end each repeats, but whose middle changes. We also remove any patterns whose repetition contains any notes from its original occurrence. Starting from the top of this sorted list, a pattern is valid when no notes of that pattern have appeared in a previous valid pattern.

(Onset, pitch)	(0.0, 72)	(0.25, 74)	(0.5, 76)	(0.75,72)	(1.0, 72)	(1.25, 74)	(1.5, 76)	(1.75, 72)
(0.0, 72)	-	-	-	-	-	-	-	-
(0.25, 74)	(0.25, 2)	-	-	-	-	-	-	-
(0.5, 76)	(0.5, 4)	(0.25, 2)	-	-	-	-	-	-
(0.75, 72)	(0.75, 0)	(0.5, -2)	(0.25, -4)	-	-	-	-	-
(1.0, 72)	(1.0, 0)	(0.75, -2)	(0.5, -4)	(0.25, 0)	-	-	-	-
(1.25, 74)	(1.25, 2)	(1.0, 0)	(0.75, -2)	(0.5, 2)	(0.25, 2)	-	-	-
(1.5, 76)	(1.5, 4)	(1.25, 2)	(1.0, 0)	(0.75, 4)	(0.5, 4)	(0.25, 2)	-	-
(1.75, 72)	(1.75, 0)	(1.5, -2)	(1.25, -4)	(1.0, 0)	(0.75, 0)	(0.5, -2)	(0.25, -4)	-

Table 2: Translation vectors of the first two bars of Figure 1. Colors signify identical translation vectors (potential patterns), and uncoloured, non-empty cells are unique. The pitches are indicated by their MIDI note number.

So, in this example, the first four notes (corresponding to the columns of the purple translation vectors in the table) can be shifted by four beats to obtain the last four notes (corresponding to the rows of the purple vectors in the table), and are saved as a valid pattern.

3.3 Special cases

For both the string-based and translation-based algorithms, when a note is not part of any larger pattern, it is considered as a pattern itself, making it easier for the generation. In the case where the last pattern of the prime is unique, the probabilities for the next pattern are set to the unigram probability of each pattern appearing in the prime, regardless of position.

3.4 Generation

Once the patterns are found, both algorithms work in the same way: they transform the sequence of notes into a sequence of patterns, and apply a first-order Markov model on this transformed sequence to generate, not a continuation of notes, but rather a continuation of patterns. Then, we translate this sequence of patterns back into a sequence of midi notes: a list of tuples with pitch, onset, duration and velocity (which we always set to 100).

3.5 Smoothing

These algorithms as presented can only produce transitions that already exist in the given prime. In order to inject additional creativity into the generation, we apply a form of additive smoothing as follows. First, we produce the transition matrix over the patterns found in the prime (as shown in Table 1 for pitches). Then, each probability is multiplied by some number $\alpha < 1$, thus removing some probability mass from the transitions found in the prime. We then distribute the remaining $1 - \alpha$ probability mass among the states, proportional to the normalised histogram (the unigram probability of each state; shown in Table 3 for “Frère Jacques”). In our experiments, we set α to 0.9.

This approach ensures that states that often appear in the prime also appear more often than others in the generation, but would still create transitions that don’t exist in the prime, resulting in some “creativity”. With this smoothing, the transition matrix shown in Table 1 is changed as in Table 4.

C	D	E	F	G
0.2857	0.1429	0.2857	0.1429	0.1429

Table 3: The normalised histogram (unigram probabilities) of the pitches of Figure 1.

Pitch class	C	D	E	F	G
C	0.25357	0.46429	0.25357	0.01429	0.01429
D	0.02857	0.01429	0.92857	0.01429	0.01429
E	0.47857	0.01429	0.02857	0.46429	0.01429
F	0.02857	0.01429	0.02857	0.01429	0.91429
G	0.02857	0.01429	0.92857	0.01429	0.01429

Table 4: Transition matrix of Figure 1 with additive smoothing ($\alpha = 0.9$).

4. EVALUATION

4.1 Baseline Methods

We compare against two different baseline methods, neither of which uses any sort of pattern detection or explicit repetition generation. The first, *Baseline*, is based on the model described in [17], and is also used as a baseline in the MIREX PP task. It is a first-order Markov model which is trained on and outputs (inter-onset-interval, pitch) tuples. The second, *Simple*, outputs the combination of two first-order Markov models: one which outputs the pitch of each note and a second which outputs the inter-onset interval (IOI) for each.

4.2 Metrics

For a quantitative evaluation, we apply the two metrics also used in the MIREX PP task: cardinality score and pitch score. [4]

4.2.1 *Cardinality score*

The cardinality score (CS) is defined as:

$$\mathrm{CS}(\mathbf{P}, \mathbf{Q}) = \max_{t \in \mathbf{T}} |\{q \, \forall \, q \in \mathbf{Q} \mid (q + t) \in \mathbf{P}\}| \qquad (2)$$

Here, **P** and **Q** sets of (onset, pitch) tuples for the true and generated continuations, respectively, and **T** is the set of all possible translation vectors that make a note from **Q**

[4] The code used for evaluation is available at `https://github.com/BeritJanssen/PatternsForPrediction`.

overlap a note from $\mathbf{P}$, formally defined as:

$$\mathbf{T} = \{p - q \,\forall\, p \in \mathbf{P}, q \in \mathbf{Q}\} \tag{3}$$

In other words, a higher score reflects how similar two continuation are in their general shape, ignoring shifts in time and pitch. Using this score, recall and precision can be calculated as in Equations 4 and 5 (note that we subtract 1 from each numerator and denominator since at least one note is guaranteed to overlap), and F1 is calculated as their harmonic mean as usual. Intuitively, recall is the proportion of the continuation which has been correctly generated, and precision is the proportion of the generation that matches the continuation.

$$Recall = \frac{\mathrm{CS}(\mathbf{P}, \mathbf{Q}) - 1}{|\mathbf{P}| - 1} \tag{4}$$

$$Precision = \frac{\mathrm{CS}(\mathbf{P}, \mathbf{Q}) - 1}{|\mathbf{Q}| - 1} \tag{5}$$

The cardinality scores are also plotted as a function of beats after the prime's final onset position, where each CS value considers only notes from the true continuation up to that position. So, if a generation contains a contour that matches the beginning of the true continuation, but not the end (as might be expected), that generation's CS will be higher for smaller onset values (after some possible initial values of 0 corresponding to positions before the 2nd generated note). The CS at onset 10 corresponds to the overall cardinality score with respect to 10 beats after the final note from the prime, and thus represents the best indicator of a generation's over quality in this regard.

4.2.2 *Pitch score*

One issue with the cardinality score is that it is pitch invariant: if the generated continuation is equal to the true continuation but shifted vertically, it would get a perfect score of 1, since the whole sequence of notes can be translated into the true continuation with only one translation vector. Pitch score is used to solve this problem. It is the magnitude of the overlap between normalised histograms of the true and generated continuations (an example of a normalised histogram is given in Table 3). This process is repeated disregarding octaves (i.e., taking each pitch modulo 12).

4.3 Data

The dataset used for evaluation is composed of two parts: the prime, from which a continuation will be generated, and the true continuation, which will be compared with the outputs of the different generation systems. Specifically, we use the large monophonic dataset prepared for the MIREX PP task, which consists of excerpts taken from the Lakh MIDI dataset [24].

4.4 Quantitative Results

Each system's CS with respect to onset position are plotted in Figure 2. We can first observe that the simple first-order

Model	**mean**	**median**	**std**
Baseline [17]	0.542	0.555	0.211
Simple	0.544	0.556	0.206
String-based	0.553	0.565	0.220
Translation-based	0.574	0.588	0.218

Table 5: Pitch scores.

Model	**mean**	**median**	**std**
Baseline [17]	0.616	0.635	0.188
Simple	0.618	0.633	0.185
String-based	0.627	0.650	0.196
Translation-based	0.647	0.670	0.192

Table 6: Modulo 12 pitch scores.

Markov model performs poorly, which is somewhat expected. The next best model appears to be the baseline, and both of our systems with pattern recognition achieve an improvement in precision, recall, and F1, with the translation-based system performing the best. Our systems see the greatest improvement in terms of recall, which suggests that they output more notes of the correct general shape. Our systems also avoid the sharp decrease in performance of the baseline system over the first few beats, instead seeing a slow decrease across the duration of the continuation. This makes sense conceptually, because instead of generating one *note* at a time, our systems generate one *pattern* at a time. Thus, they are typically fewer steps along in the generation process after any given duration (each step holds a potential for the generation to go off track).

Table 5 shows each system's pitch scores (both with and without octave equivalence) in violin plots, and the exact values are given in Tables 5 and 6. We can observe that the systems are quite similar. However, the translation-based system again achieves the best results, though only marginally in this case. The scores for all systems are only slightly above 0.5, which shows that the majority of generated notes are of the correct pitch, but there are still many incorrect notes from this perspective. Of these errors, fewer than 10% are simple octave errors (this can be measured by the difference between the values in Tables 5 and 6).

4.5 Examples

For a more in-depth comparison of the performance of our approaches to pattern-based generation, we now present an in-depth analysis of the translation-based, string-based, and simple (no pattern) outputs for two example primes. In all figures in this section, the string-based and translation-based patterns are annotated with red and green brackets, respectively.

The first example is a prime from our test dataset, shown in Figure 4. From the red and green annotations, it is clear that the translation-based pattern detection algorithm has found longer patterns on average than the string-based one. In particular, there are many more single-note "patterns" for the string-based algorithm. The true continuation, as well as the continuation generated by each system, are

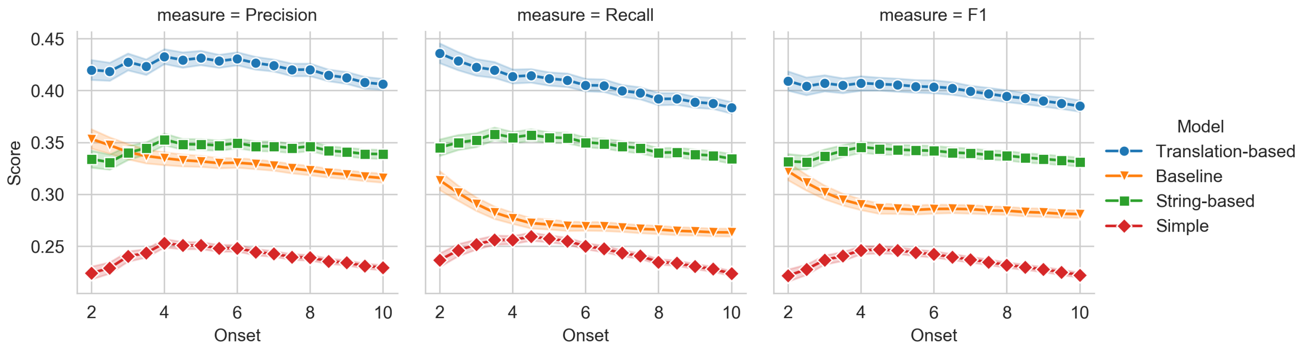


Figure 2. Cardinality scores for our proposed systems (Transition-based and String-based) as well as the simple Markov model and the baseline. The x-axis represents onset position in the true continuation, measured in quarter notes.

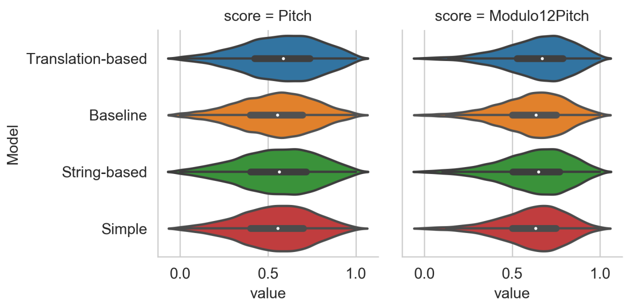


Figure 3. Pitch scores for our three models and the baseline.

shown in Figure 6 (note that the continuations are sampled from distributions, so will change each time). We can observe that the generation without pattern recognition is the least similar to the true continuation. The string-based generation is better: it has the correct rhythm, and even nearly the correct pitch contour for the first half (though notes are shifted up by both minor and major thirds). However, the translation-based generation matches the true continuation exactly.

Clearly, the translation-based approach performs quite well when repetition is present in the prime. However, that is also its drawback: its generations heavily depend on the structure of the prime: if the prime is not repetitive, it will not be able to rely on such pattern generations to produce, and the generation could be poor.

The example prime shown in Figure 5 (the "Castle in the sky" theme, composed by Joe Hisaishi) is one such case. Notice how many small patterns are found by both methods, and how short each one is. The generations (shown in Figure 7) reflect this: only very short patterns can be generated, and none of the generations match the true continuation very well. The string-based generation is slightly better in terms of rhythm, at least matching the dotted-half note, quarter note rhythm in bars two and four, as well as the position of the dotted-quarter notes in bars one and three. In terms of pitch, none of the generations perform very well, although they produce a reasonable set of notes.

So, it can be seen that the translation-based method produces the most accurate generations for repetitive primes, but falls back to around the performance of the less sophisticated systems for primes without much repetition.

5. CONCLUSION

In this work, we presented two novel systems for generating music with explicit repetition, based on a given prime. The systems generally work by first detecting repeated patterns in the prime, and using these to inform the generation process with a simple Markov model. We show that a more flexible, translation-based pattern detection algorithm is able to capture more sophisticated forms of repetition, which improves its generations. Overall, this pattern-based approach works well when the prime is somewhat repetitive; however, it can struggle otherwise.

This reliance on patterns is another potential drawback of our system in that it has no mechanism to make small changes to the found patterns. The creativity involved in making small changes to repeated patterns throughout a piece of music is very important to such repetition, and our system currently lacks this ability. Future work could try to improve this in two ways. First, during pattern detection, the algorithm could be adapted to find such patterns with variations, explicitly noting the types of variations seen in the prime. Then, during generation, the model could explicitly add some of these or other variations into the generated patterns. This would allow the system to produce more creative generations, while still ensuring that it has some repetitive structure. The evaluation of music generation is a very difficult problem, and in future work, we could also include a subjective evaluation involving a group of experts, especially when using primes that do not show repeated patterns (since these generations can be the most varied).

In this work, we concentrated on monophonic music, but similar algorithms for polyphonic music can also be developed in a few ways. The simplest is to apply a voice separation model (e.g., [25]) as a preprocessing step, then producing one generation per voice. Another option is to enlarge the state-space of the Markov model ton include combinations of notes, although this adds a significant amount of complication to the process.

Figure 4. MIDI sample taken from the MIREX 2019: Patterns for prediction dataset. Red and green brackets show the patterns found by the string-based and translation-based algorithms, respectively.

Figure 5. "Castle in the sky" theme, composed by Joe Hisaishi. In red, the patterns found by the string-based approach, in green, the ones found by the translation-based algorithm.

(a) True continuation.

(b) No pattern recognition.

(c) String-based pattern recognition (patterns in red brackets).

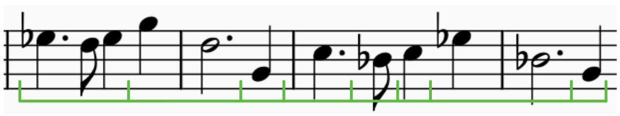
(d) Translation-based pattern recognition (patterns in green brackets).

Figure 6. Generated and true continuations of the prime shown in Figure 4. Exact onset timing has been quantized to the nearest 16th note.

(a) True continuation.

(b) No pattern recognition.

(c) String-based pattern recognition (patterns in red brackets).



(d) Translation-based pattern recognition (patterns in green brackets).

Figure 7. Generated and true continuations of the prime shown in Figure 5. Exact onset timing has been quantized to the nearest 16th note.

The code for this work is available online [5].

Acknowledgments

Research supported through the Swiss National Science Foundation within the project "Distant Listening – The Development of Harmony over Three Centuries (1700–2000)" (Grant no. 182811).

6. REFERENCES

[1] A. Ockelford, *Repetition in music: Theoretical and metatheoretical perspectives*. Routledge, 2017.

[2] W. E. Caplin, *Classical form: A theory of formal functions for the instrumental music of Haydn, Mozart, and Beethoven*. Oxford University Press, 2001.

[3] D. Huron, *Sweet anticipation: Music and the psychology of expectation*. MIT press, 2008.

[4] E. H. Margulis, *On repeat: How music plays the mind*. Oxford University Press, 2014.

[5] B. L. Sturm and O. Ben-Tal, "Taking the models back to music practice: Evaluating generative transcription models built using deep learning," *Journal of Creative Music Systems*, vol. 2, no. 1, 2017.

[6] B. Janssen, P. van Kranenburg, and A. Volk, "Finding occurrences of melodic segments in folk songs employing symbolic similarity measures," *Journal of New Music Research*, vol. 46, no. 2, pp. 118–134, 2017.

[7] G. Shibata, R. Nishikimi, E. Nakamura, and K. Yoshii, "Statistical music structure analysis based on a homogeneity-, repetitiveness-, and regularity-aware hierarchical hidden semi-markov model." in *ISMIR*, 2019, pp. 268–275.

[8] B. Sturm, J. F. Santos, and I. Korshunova, "Folk music style modelling by recurrent neural networks with long short term memory units," in *16th International Society for Music Information Retrieval Conference*, 2015.

[9] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, "Deep learning techniques for music generation – a survey," 2019.

[10] L.-C. Yang, S.-Y. Chou, and Y. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation," in *International Society for Music Information Retrieval Conference*, 2017.

[11] Z. Wang, Y. Zhang, Y. Zhang, J. Jiang, R. Yang, J. Zhao, and G. Xia, "Pianotree VAE: Structured representation learning for polyphonic music," in *International Society for Music Information Retrieval Conference*, 2020.

[12] D. Conklin, "Prediction and entropy of music," Master's thesis, The University of Calgary, 1992.

[13] M. T. Pearce, "The construction and evaluation of statistical models of melodic structure in music perception and composition," Ph.D. dissertation, City University London, 2005.

[14] S. Lattner, M. Grachten, and G. Widmer, "Imposing higher-level structure in polyphonic music generation using convolutional restricted boltzmann machines and constraints," *Journal of Creative Music Systems*, vol. 2, pp. 1–31, 2018.

[15] D. Meredith, K. Lemström, and G. A. Wiggins, "Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music," *Journal of New Music Research*, vol. 31, no. 4, pp. 321–345, 2002.

[16] P.-Y. Rolland, "Discovering patterns in musical sequences," *Journal of New Music Research*, vol. 28, no. 4, pp. 334–350, 1999.

[17] T. Collins, R. Laney, A. Willis, and P. H. Garthwaite, "Chopin, mazurkas and markov," *Significance*, vol. 8, no. 4, pp. 154–159, 2011.

[18] F. Pachet, "The continuator: Musical interaction with style," *Journal of New Music Research*, vol. 32, no. 3, pp. 333–341, 2003.

[19] V. Velardo, M. Vallati, and S. Jan, "Symbolic melodic similarity: State of the art and future challenges," *Computer Music Journal*, vol. 40, no. 2, pp. 70–83, 06 2016.

[20] E. Cambouropoulos, "Musical parallelism and melodic segmentation:: A computational approach," *Music Perception*, vol. 23, no. 3, pp. 249–268, 2006.

[21] D. Meredith, "Cosiatec and siateccompress: Pattern discovery by geometric compression," in *International society for music information retrieval conference*. International Society for Music Information Retrieval, 2013.

[22] I. Y. Ren, H. V. Koops, A. Volk, and W. Swierstra, "In search of the consensus among musical pattern discovery algorithms," in *Proceedings of the 18th International Society for Music Information Retrieval Conference*. ISMIR press, 2017, pp. 671–678.

[23] P. Boot, A. Volk, and W. B. de Haas, "Evaluating the role of repeated patterns in folk song classification and compression," *Journal of New Music Research*, vol. 45, no. 3, pp. 223–238, 2016.

[24] C. Raffel, "Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching," Ph.D. dissertation, Columbia University, 2016.

[25] A. McLeod and M. Steedman, "HMM-based voice separation of MIDI performance," *Journal of New Music Research*, vol. 45, no. 1, pp. 17–26, 2016.

[5] https://github.com/v1ruso/BachelorProject