GENERALIZED SKIPGRAMS FOR PATTERN DISCOVERY IN POLYPHONIC STREAMS

Christoph Finkensiep **Markus Neuwirth** **Martin Rohrmeier**
École Polytechnique Fédérale de Lausanne
{christoph.finkensiep,markus.neuwirth,martin.rohrmeier}@epfl.ch

ABSTRACT

The discovery of patterns using a minimal set of assumptions constitutes a central challenge in the modeling of polyphonic music and complex streams in general. Skipgrams have been found to be a powerful model for capturing semi-local dependencies in sequences of entities when dependencies may not be directly adjacent (see, for instance, the problems of modeling sequences of words or letters in computational linguistics). Since common skipgrams define locality based on indices, they can only be applied to a single stream of non-overlapping entities. This paper proposes a generalized skipgram model that allows arbitrary cost functions (defining locality), efficient filtering, recursive application (skipgrams over skipgrams), and memory efficient streaming. Further, a sampling mechanism is proposed that flexibly controls runtime and output size. These generalizations and optimizations make it possible to employ skipgrams for the discovery of repeated patterns of close, nonsimultaneous events or notes. The extensions to the skipgram model provided here do not only apply to musical notes but to any list of entities that is monotonic with respect to a given cost function.

1. INTRODUCTION

Discovering relevant patterns in a given corpus of musical pieces is a central problem for music modeling and music information retrieval (MIR) and is crucial for a range of applications from search to stylistic modeling. While there exist many approaches for modeling monophonic melodies [7, 6], polyphony constitutes a persistent challenge due to the vast amount of latent structural patterns that occur on multiple levels. These patterns involve surface ornaments and accompaniment figurations, contrapuntal configurations, latent polyphony comprising multiple interleaved voices, and harmonic and voice-leading schemata.

While many of the underlying patterns are themselves relatively simple, identifying these patterns is challenging, because it involves distinguishing the relevant notes while ignoring others. In addition, many patterns do not specify notes exactly but leave some flexibility when being instantiated, especially concerning timing. Finally, multiple patterns may co-occur simultaneously or in an interleaved manner.

When modeling the latent structure of polyphony, it is important to find the characteristic properties of the structure to be modeled. Therefore, it is advantageous to start from a model with minimal assumptions about the target structure and add assumptions to the basic model until the desired patterns are found. This way, the properties of the modeled structure are always clear and well-separated from the assumptions inherent in the underlying representation.

There are a variety of methods for modeling sequential data with minimal assumptions, such as those developed in computer linguistics, that treat the data as a single stream of events. However, these cannot be straightforwardly applied to polyphonic data without adding further implicit assumptions or removing information contained in the original data. Therefore, a generalization of skipgrams [4] is developed in this paper that is applicable to a stream of polyphonic notes that need not be explicitly presented as separate voices. This model is applied to a musical corpus for the discovery of polyphonic patterns.

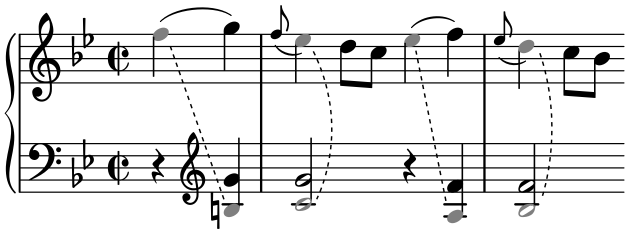
In the remainder of this paper, we first discuss related work in more detail (Section 2); we then describe our approach for generalized skipgrams (Section 3); finally, we describe and discuss our empirical evaluations (Sections 4 and 5).

2. RELATED WORK

Among polyphonic structures, *voice-leading schemata* are particularly prominent in recent research [3]. Schemata can be understood as structural building blocks that can be elaborated in multiple ways. They are defined as fixed patterns of two to four voices where the soprano and bass constitute specific patterns relative to the key of the piece (or a segment within that piece) and may be supplemented with one or two middle voices. The core challenge from an MIR perspective is that the structural elements in each stage of the sequence need not occur simultaneously, owing to highly flexible note elaborations. Thus, instances of schemata in the music are "semi-local". An example of this problem can be seen in Figure 1. The underlying schema consists of four stages and is elaborated by neighbor and

$$\begin{tikzcd}
\hat{5} \arrow[r] \arrow[d, dash, dotted] & \hat{4} \arrow[r] \arrow[d, dash, dotted] & \hat{4} \arrow[r] \arrow[d, dash, dotted] & \hat{3} \arrow[d, dash, dotted] \\
\#\hat{1} \arrow[r] & \hat{2} \arrow[r] & \hat{7} \arrow[r] & \hat{1}
\end{tikzcd}$$

(a) The "Fonte" schema as characterized by a typical outer-voice movement in scale degrees



(b) A realization of the Fonte in a piece

Figure 1: An example of a voice-leading schema

passing notes. The first stage consists of non-overlapping notes, so there is no point in time where both notes sound together. The same applies for the third stage.

This semi-locality property of schema patterns can be met by formalizing an extension of the *skipgram* formalism, which has been successfully applied in linguistics to sequences of words or letters (for a review see [4]). Skipgrams in the original version can only be applied to "monophonic" streams like text, melodies or slices of polyphonic music, as has been done in [8]. The generalized version of skipgrams as proposed in the present paper allows not only the application to truly polyphonic streams but also recursive application, which can be used to build nested structures like schema patterns.

Previously, polyphonic music was mainly modeled using slicing techniques, i.e., cutting the piece vertically at each note onset or offset. In [8], common, index-based skipgrams as well as an onset-time-based variant are applied to slices reduced to a "voice-leading type" representation, similar to the representation used here (for more details see Section 4.2). The approach presented in this paper takes the idea two steps further by generalizing the cost function (allowing non-slice representations as input) and by building even the vertical structure with this generalized skipgram method, in addition to the horizontal structure.

Multiple viewpoint systems (e.g., [1, 2, 10, 9]) take a sequence of slices and derive sequence features, or "viewpoints", from it. Polyphonic structure is modeled by including information about the continuation of notes across slices. For prediction, n-grams of all lengths are combined by comparing all suffixes of a given gram to other grams of the corresponding length. However, slicing techniques are generally problematic when grouping non-overlapping notes, as these are not contained in any single slice.

An alternative to slices is suggested in [5] where polyphonic music is encoded as a set of data points in a multidimensional space. Accordingly, patterns are orthogonal projections (i.e., considering only some features, not all) of subsets of the data points that can be translated to a different position (in both pitch and time). This translation operation, however, permits only exact matches in the selected dimensions and cannot account for temporally varied patterns.

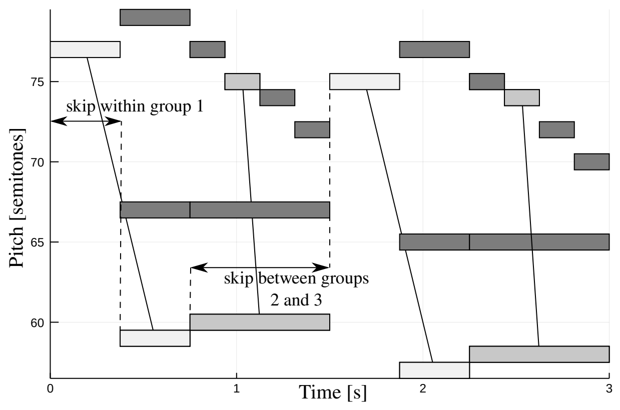


Figure 2: An example of applying skipgrams to polyphonic music displayed in a piano-roll visualization. The highlighted notes are members of the skipgram, the stages are indicated by solid lines between notes belonging to the same stage. The skip cost within and between stages is given by inter-onset intervals. This 2×4 skipgram represents the same pattern as the polyphonic schema in Figure 1.

3. GRAM-BASED METHODS AND GENERALIZED SKIPGRAMS

Gram-based methods extract short sequences of entities from a longer stream of entities (e.g., words, letters, or notes). The most basic gram model, the n-gram, is just a consecutive subsequence in the input stream that has n elements. Skipgrams extend the n-gram idea by allowing non-adjacent subsequences that "skip" up to k elements [4].

Both n-grams and skipgrams assume that the distance between entities is determined by their *position in the stream*. While this assumption might be reasonable for text, it is problematic for other applications that involve general temporal streams, in particular streams of musical events such as notes. Therefore, it is desirable to measure the distance between events (notes) based on their timing information, i.e., onset, offset, and duration. Second, while notes might be simultaneous in a score, they occur sequentially in a stream or list-of-notes representation, which becomes problematic if distance is measured by index.

Sears et al. [8] avoid the latter problem by operating on a slice representation of a piece, in which slices have a unique onset and do not overlap. They partially solve the first problem by replacing the maximal number of skips with a maximal inter-onset-interval, i.e., a time-based distance measure. Our paper presents a generalization of skipgrams to arbitrary pairwise cost functions for streams that are monotonic with respect to the cost function, which allows both efficient implementation and streams of overlapping entities.

Consider Figure 2, a piano-roll representation of a polyphonic piece of music. The notes that make up a single stage of a voice-leading schema might not be simultaneous, but should be close together. Given the notes of the piece as a list of triples $(onset, offset, pitch)$, candidates for a schema stage can be found by considering all groups of notes (pairs, in the case of two voices) that lie within a certain distance. In the traditional skipgram approach, this distance would be measured by the index of each note in the list. However, in the case of voice-leading schemata, it is more meaningful to measure this distance with respect to the timing of the notes, e.g., as the distance between onsets or the distance between the offset of the earlier and the onset of the latter note.

Since a voice-leading schema consists of several consecutive stages, it is natural to apply this more general idea of skipgrams again, now to the list possible stages. As with notes, the distance between two stages can be defined in several ways, e.g., as the distance from the beginning of the first stage to the beginning of the second, or the amount of time between the stages. In the following section, an algorithm that enumerates skipgrams over streams of arbitrary objects for arbitrary definitions of distance is presented along with some useful extensions.

3.1 The Generalized Skipgram Algorithm

The basic algorithm for generalized skipgrams is shown in Algorithm 1. It takes a stream of objects (e.g., notes or schema stages), an upper bound on the allowed "skip" k, the length of the generated skipgrams n, and a cost function c. The cost function is used to represent the distance between two objects: the combined cost across a skipgram must not be greater than k. While it traverses the input stream, a set of prefixes (incomplete skipgrams) is maintained. For each prefix that can be extended by the current element without increasing the total cost beyond k, the extended version is added to the prefix set. Prefixes of length n are added to the output and removed from the set of prefixes. Finally, the current element starts a new prefix.

In order to keep the set of prefixes small, a prefix is removed as soon as extending it increases the cost beyond k. As long as the stream is sorted in a way that every subsequent element would increase the cost of the prefix even further, this optimization does not discard valid skipgrams. That means the input stream $input$ must satisfy

$$\forall x < y < z \in input : c(x, y) \leq c(x, z),$$

where $x < y$ denotes that x appears before y in $input$.

The cost function can handle the distance in several ways. For example, if the distance between the first and the last note in a skipgram should be limited, then the cost equals the distance between the onsets of two neighboring notes in the skipgram. On the other hand, if the distance between two neighboring notes is to be limited, the cost can be defined non-continuously as 0 if the notes are within the allowed distance and $k + 1$. This way, the combined cost is 0, except when a single neighbor pair of notes is too far apart, in which case it exceeds k.

Algorithm 1 The basic algorithm for enumerating generalized skipgrams.

```
1:  function SKIPGRAMS($input, k, n, c$)
2:      $pfxs \leftarrow \{\}$
3:      $output \leftarrow [\,]$
4:      $\mathrm{cost}(p, x) = \sum_{i=1}^{l-1} c(p_i, p_{i+1}) + c(p_l, x)$
5:      for $x \leftarrow input$ do
6:          $open \leftarrow \{p \mid p \in pfxs, \mathrm{cost}(p, x) \leq k\}$
7:          $ext \leftarrow \{p \circ x \mid p \in open\}$
8:          $append(output, [p \mid p \in ext, |p| = n])$
9:          $pfxs \leftarrow open \cup \{p \mid p \in ext, |p| < n\} \cup \{x\}$
10:     end for
11:     return $output$
12: end function
```

Note that the skipgram algorithm traverses the input stream exactly once, so streaming it is straightforward. Similarly, the output can be streamed instead of collected, either using some form of concurrency and a channel between the output of the skipgram generator and some consumer, or non-concurrently using an iterator pattern.

3.2 Early Filtering

If the list of generated skipgrams will be filtered for some property (e.g., only selecting notes that do not overlap or that match a given schema prototype), it is desirable to filter out prefixes that cannot be completed to satisfy the predicate as early as possible, instead of generating all of its completions first. Therefore, an extension of Algorithm 1 additionally takes a predicate $pred$, which it applies to every generated prefix. Every prefix that does not satisfy $pred$ is removed. As with the cost function, this predicate can be defined freely.

3.3 Stable Ordering

Algorithm 1 adds its output in the order in which prefixes are completed. As a consequence, the output stream does not retain the order of the first element in each skipgram with respect to the input order. Instead the input order is reflected in the last element of each skipgram.

For some applications, it might be desirable to keep the order of the first elements intact. For example, when computing skipgrams of skipgrams (e.g., first groups of notes, then sequences of groups), the second skipgram pass expects its input to be monotonic with respect to the cost function. In the case of note groups, this will likely depend on the earliest onset in the group. The first skipgram pass takes a list of notes ordered by onset, but the note groups it returns will not be ordered by the onset of their first (and therefore earliest) note but by the onset of their last note.

In general, the appropriate order of skipgrams can be obtained by generating the whole list of skipgrams and sorting it, but this would destroy the streaming property of the algorithm. As the number of skipgrams grows quickly, it might not even be feasible to keep all skipgrams in memory. Furthermore, as k is intended to limit the range of skipgrams, a skipgram can only be displaced as much as k, so the array will almost be sorted, and sorting can be done on the fly.

Stable ordering can be ensured efficiently by holding back completed skipgrams (instead of adding them to the output as soon as they are generated) until no new skipgrams can be generated that should precede them. This can be achieved by using a priority queue to hold the finished but not yet released skipgrams in the correct order. In each iteration, the open prefixes are searched for the "oldest" first element. Then, all skipgrams in the output queue starting with an element not younger than this oldest initial prefix element are released. If the output needs to be ordered lexicographically, the queue content must be compared not to the oldest initial element but to the complete lexicographically oldest open prefix. The queue can be updated efficiently by sorting the newly generated prefixes and merging the resulting list with the existing queue as in a merge sort.

3.4 Sampling Skipgrams

The number of generated skipgrams as well as the asymptotic runtime of the algorithm are difficult to estimate, as they depend on the number of elements within a range of k or the number of currently open prefixes, respectively, at any point in the stream. This, in turn, depends on the combination of input and cost function, so no general statement about runtime and space complexity can be made without knowing both. In the worst case, generalized skipgrams consist of all subsets of length n from the list of L entities, generating $\binom{L}{n}$ skipgrams.

Because this amount of skipgrams is costly to enumerate and process, an alternative is to uniformly draw samples from the list of all skipgrams. For a given probability p, a biased coin is tossed for each skipgram, which determines whether the skipgram is selected or not. If this is done after the skipgram is completely generated, all skipgrams must be enumerated once, so computation time is saved only during consumption but not production. Conversely, one could toss the coin for each new prefix of length 1. This way, all extensions of a discarded prefix need not be computed, which saves computation time but also removes a whole family of related skipgrams from the output.

A third approach combines the other two by tossing a coin each time a prefix is extended. As this happens $n-1$ times for a prefix of length n (not counting the creation of the initial length 1 prefix), so the coin is biased not with p but with $p' = \sqrt[n-1]{p}$. This way, a skipgram is only included if all of its prefix extensions succeed, i.e., with probability $(p')^{n-1} = p$. Furthermore, computation time can be saved by discarding short prefixes, while variety is preserved by also discarding prefixes in later stages.

With this method, it is not possible to uniformly draw a fixed number of samples. However, the expected number of samples can be estimated by choosing a small p and extrapolating the resulting amount of sampled skipgrams. The total number of skipgrams N can be estimated similarly, as the number of sampled skipgrams is expected to be Np.

4. SKIPGRAMS ON POLYPHONIC MUSIC

4.1 Dataset

The described method is applied to 17 piano sonatas by Wolfgang Amadeus Mozart in MIDI format.[1] A schema has 2 or 3 voices and consists of 2 to 4 stages. These *dimensions* of a schema are notated as voices $\times$ stages or $n_v \times n_s$. and the sampling parameters p_v and p_s are adapted to these dimensions. The skip limit within a stage k_v is one bar in total, the limit between the stages k_s is also one bar, but per pair of stages.

4.2 Method

For the discovery of musical schemata, three assumptions are made. First, schemata are semi-local structures, that is, they extend over a limited range of time. This property is inherent in the skipgram formalism with an appropriate cost function. Second, they consist of a horizontal sequence of pseudo-vertical structures, which consist of a fixed number of possibly non-simultaneous or even non-overlapping notes. Third, patterns are characterized by their pitch content, not by their temporal properties. This pitch content is subject to certain equivalences, e.g., regarding transposition of the whole pattern or the exact octave of each pitch.

In order to find vertical structures in polyphonic pieces, skipgrams can be applied to a stream of notes. Each note has a pitch, an onset and an offset, and the notes are sorted by their onsets. For this purpose, the cost function is the difference between the onsets of two notes in the skipgram

$$c_v(n_1, n_2) = \text{onset}(n_2) - \text{onset}(n_1),$$

which in sum amounts to the onset difference between the earliest and the latest note in the group. This allows notes to overlap but restricts their temporal distance. The stable variant of the skipgram algorithm is used to ensure that the output stream is again ordered by (earliest) onset. The number of notes n_v in the first pass can be regarded as the number of voices in the vertical structure.

A second skipgram pass builds length n_s sequences of vertical structures by taking the output of the first pass as the new input. Since these structures should be horizontally organized, temporal overlap between their stages is forbidden (by defining an appropriate *pred*). The amount of time between the onsets of slices is restricted by a step function that admits a skip up to k_s for each pair of neighbors:

$$c_s(s_1, s_2) = \begin{cases} 0 & \text{if onset}(s_{2,1}) - \text{onset}(s_{1,1}) \leq k_s \\ k_s + 1 & \text{otherwise.} \end{cases}$$

Due to the large amount of skipgrams generated, the sampling parameters are adapted to the number of voices (p_v)

[1] Encoded by Craig Sapp in Kern format and converted to MIDI. Available at `https://github.com/craigsapp/mozart-piano-sonatas`.

n_v	n_s	p_s	p_v	sampled	total	cov
2	2	1.0	1.0	$3.30 \cdot 10^8$	10^8	1.0
2	3	1.0	0.001	$9.92 \cdot 10^7$	10^{11}	0.99998
2	4	1.0	10^{-6}	$3.77 \cdot 10^7$	10^{13}	0.30
3	2	0.1	1.0	$3.53 \cdot 10^8$	10^{10}	0.9997

Table 1: The combinations of parameters used to generate the results. For each combination, the sampled and the rough estimated total number of skipgrams, as well as the coverage are given. Coverage is the ratio of the number of skipgram classes encountered and the number of possible classes for the given dimensions ($12^{n_v n_s - 1}$). [2]

and stages (p_s). An example of such a nested skipgram is shown in Figure 2.

The pitch content of the resulting structures is summarized by summing the occurrences of skipgrams with similar pitch content ("skipgram classes"). Pitch combinations are grouped by sorting the pitches in each stage in ascending order, removing octave information (pitch classes), and transposing every pitch class relative to the lowest note of the first stage. For example, the sequence $(f, c', a') \rightarrow (e, c', g')$ would be encoded as $(0, 7, 4) \rightarrow (11, 7, 2)$. This is similar but not identical to the voice-leading type representation used in [8], which additionally removes the order of the pitches and the magnitude of each pitch class. Since the focus here is on polyphonic voice-leading schemata, both order and magnitude are retained. For n_v voices and n_s stages, there are $12^{n_v n_s - 1}$ possible skipgram classes, as the transposition step always sets the first bass note to 0.

5. RESULTS AND DISCUSSION

5.1 Results

Table 1 gives an overview of the used parameter combinations. For each set of parameters, it shows the number of generated skipgrams, the estimated number of total skipgrams, as well as the coverage, which is the ratio of the number of encountered skipgram classes and the number of possible classes for the given dimensions. Good coverage is important for prediction tasks, where the *zero-frequency problem* occurs (i.e., where a prediction context has not been encountered at least once, see [8]).

As the outer-voice movement is most characteristic of voice-leading schemata, the most general insight can be provided by two-voiced skipgram classes, which also have a good coverage for reasonable computational effort. Additionally, the 3×2 skipgrams were generated to observe the effect of increasing the number of voices on the found patterns. Larger dimensions did not produce a sufficient coverage due to computational constraints. The total number of skipgrams without sampling can be estimated by multiplying the sampled skipgrams with $p_s(p_v^{n_s})$. p_v needs to be exponentiated, as it factors in the output probability of a skipgram on each of its stages.

[2] Runtimes ranged from a few minutes to several hours. While parallelization was straightforward by splitting the dataset, memory usage was problematic and prevented generating skipgrams with larger dimensions.

	class	abs	rel
1	$(0,3) \rightarrow (0,2) \rightarrow (10,2) \rightarrow (10,0)$	1190	0.32
2	$(0,3) \rightarrow (3,3) \rightarrow (0,3) \rightarrow (3,3)$	1029	0.27
3	$(0,1) \rightarrow (10,1) \rightarrow (8,0) \rightarrow (8,10)$	1009	0.27
4	$(0,0) \rightarrow (5,0) \rightarrow (0,0) \rightarrow (5,0)$	976	0.26
5	$(0,0) \rightarrow (9,0) \rightarrow (5,0) \rightarrow (0,0)$	964	0.26
6	$(0,3) \rightarrow (3,3) \rightarrow (8,3) \rightarrow (3,3)$	937	0.25
7	$(0,1) \rightarrow (10,1) \rightarrow (0,0) \rightarrow (8,10)$	934	0.25
8	$(0,0) \rightarrow (9,0) \rightarrow (0,0) \rightarrow (9,0)$	921	0.24
9	$(0,0) \rightarrow (10,1) \rightarrow (10,0) \rightarrow (8,10)$	902	0.24
10	$(0,3) \rightarrow (0,1) \rightarrow (10,1) \rightarrow (10,0)$	897	0.24

Table 2: The 10 most frequent 2×4 skipgram classes that have no repeating stages

Table 3 shows the 10 most frequent skipgram classes found for each set of parameters. As the most frequent patterns mainly indicate arpeggiation patterns (see Section 5.2), an additional filter is applied to the 2×4 skipgrams, which forbids the repetition of a stage. The resulting 10 most frequent patterns are shown in Table 2.

In addition to enumerating (or sampling from) all skipgrams in the corpus, the generalized skipgram algorithm can be used as a pattern matcher or schema finder. Figure 2 shows the first skipgram matching a two voiced pattern $(0, 6) \rightarrow (1, 4) \rightarrow (10, 4) \rightarrow (11, 3)$ in the third movement of Mozart's third piano sonata (from which the example in Figure 1 is taken), found by enumerating the pieces 2×4 skipgrams. Pattern matching can be performed efficiently by using the early filtering mechanism described in Section 3.2 to remove prefixes that cannot match.

5.2 Discussion

As Table 1 shows, even with moderate sampling the coverage is excellent for smaller dimensions. As the dimensions increase and the sampling probability decreases, the coverage rapidly decreases as well, because the large number of possible pitch combinations conflicts with the increased need for reducing the computational effort via sampling. For example, for 2 voices and 4 stages, there are $12^7 \approx 3.58 \cdot 10^7$ possible skipgram classes, but the total number of sampled skipgrams was only $3.77 \cdot 10^7$. In principle, however, the flexibility of nested skipgrams provides a good coverage, confirming the results in [8] for flat skipgrams. For larger problem instances of up to 4×4 skipgrams, the computational problem can be solved by good parallelization and sufficient computation resources.

Based on frequency, the patterns found in the corpus are dominated by interval combinations as they appear in major and minor triads, followed by simple step-wise relations. This cannot be explained by the fact that the music of Mozart is mostly triadic to a large extent, as the stages of a single skipgram are strictly non-simultaneous. In music that is triadic but consists of sequences of nonarpeggiated, non-repeating chords, the stages of a skip-gram will be taken from different chords. Thus, the found patterns reveal the usage of harmonic *surface patterns* such as Alberti bass. This becomes especially clear from variants of $(0,0) \rightarrow (5,0)$ and $(0,0) \rightarrow (9,0)$, which are consistently ranked very high and indicate a prevalence of the fifth in combination with the third or the root of a major triad, resembling the Alberti bass pattern.

In contrast, the filtered patterns shown in Table 2 mainly consist of instances of the typical voice-leading pattern of descending 3-2 suspension sequences. This pattern is used as a typical elaboration procedure in several voice-leading schemata. This finding shows that nested skipgrams are very well capable of representing polyphonic structures. The remaining question is how to automatically distinguish surface patterns from patterns on higher structural levels in the generated skipgrams.

6. CONCLUSION

The results clearly show that the generalized skipgram formalism is capable of modeling streams of events that have a *non-flat* shape, such as streams of notes in polyphonic music. The monotonicity property explained in Section 3.1 is a general criterion for the applicability of the presented algorithm. Thus, generalized skipgrams can prove useful for a wide range of problems from various domains other than music that deal with sequential but overlapping data.

With respect to polyphonic music, generalized skipgrams provide a powerful mechanism for accessing polyphonic structure, solving the problem of building vertical structures from non-overlapping notes. Hence, a potentially powerful application of generalized skipgrams is to use them as the basic representation in variety of other methods that provide rich pattern languages, replacing the currently used sequence-of-slices structure. For example, it is possible to apply viewpoint techniques to the skipgrams generated from a polyphonic stream.

Finally, the skipgram approach requires very little assumptions on its own, but can easily be extended to filter for more advanced, theoretically or empirically motivated properties. The discovery of schema-like patterns, for example, will require to add appropriate filters that separate surface from middleground patterns. This helps to advance the understanding of the essential properties of schemata, as the added assumptions can be clearly separated from the ones inherent in the skipgram representation.

7. ACKNOWLEDGEMENTS

The research presented in this paper is generously supported by the Volkswagen Foundation and Claude Latour. We also thank the anonymous reviewers for their helpful comments.

	class	abs	rel
1	$(0,0) \rightarrow (0,0)$	$3.3e6$	10.0
2	$(0,0) \rightarrow (5,0)$	$1.38e6$	4.18
3	$(0,0) \rightarrow (0,5)$	$1.34e6$	4.06
4	$(0,0) \rightarrow (9,0)$	$1.34e6$	4.06
5	$(0,0) \rightarrow (7,0)$	$1.32e6$	4.01
6	$(0,7) \rightarrow (7,7)$	$1.31e6$	3.96
7	$(0,5) \rightarrow (0,0)$	$1.28e6$	3.89
8	$(0,3) \rightarrow (3,3)$	$1.27e6$	3.86
9	$(0,0) \rightarrow (0,7)$	$1.22e6$	3.7
10	$(0,0) \rightarrow (10,0)$	$1.21e6$	3.68
1	$(0,0) \rightarrow (0,0) \rightarrow (0,0)$	121073	1.22
2	$(0,0) \rightarrow (0,0) \rightarrow (9,0)$	44143	0.44
3	$(0,0) \rightarrow (0,0) \rightarrow (5,0)$	43764	0.44
4	$(0,0) \rightarrow (5,0) \rightarrow (0,0)$	40859	0.41
5	$(0,3) \rightarrow (3,3) \rightarrow (3,3)$	40543	0.40
6	$(0,7) \rightarrow (7,7) \rightarrow (7,7)$	39470	0.39
7	$(0,0) \rightarrow (9,0) \rightarrow (0,0)$	39056	0.39
8	$(0,0) \rightarrow (0,0) \rightarrow (10,0)$	34975	0.35
9	$(0,0) \rightarrow (0,0) \rightarrow (0,5)$	29343	0.29
10	$(0,0) \rightarrow (0,0) \rightarrow (7,0)$	28667	0.28
1	$(0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (0,0)$	6381	0.169
2	$(0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (9,0)$	2436	0.065
3	$(0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (5,0)$	2386	0.063
4	$(0,0) \rightarrow (0,0) \rightarrow (9,0) \rightarrow (0,0)$	2184	0.058
5	$(0,0) \rightarrow (0,0) \rightarrow (5,0) \rightarrow (0,0)$	2173	0.058
6	$(0,0) \rightarrow (5,0) \rightarrow (0,0) \rightarrow (0,0)$	2075	0.055
7	$(0,3) \rightarrow (3,3) \rightarrow (3,3) \rightarrow (3,3)$	2031	0.054
8	$(0,0) \rightarrow (9,0) \rightarrow (0,0) \rightarrow (0,0)$	2001	0.053
9	$(0,7) \rightarrow (7,7) \rightarrow (7,7) \rightarrow (7,7)$	1918	0.051
10	$(0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (10,0)$	1753	0.046
1	$(0,0,0) \rightarrow (0,0,0)$	461238	1.31
2	$(0,0,0) \rightarrow (9,0,0)$	214255	0.61
3	$(0,3,3) \rightarrow (3,3,3)$	208562	0.59
4	$(0,0,0) \rightarrow (5,0,0)$	205212	0.58
5	$(0,7,7) \rightarrow (7,7,7)$	200170	0.57
6	$(0,3,3) \rightarrow (0,3,3)$	172370	0.49
7	$(0,0,0) \rightarrow (10,0,0)$	162212	0.46
8	$(0,2,2) \rightarrow (2,2,2)$	133658	0.38
9	$(0,7,7) \rightarrow (0,7,7)$	131357	0.37
10	$(0,0,2) \rightarrow (0,0,0)$	128846	0.37

Table 3: The 10 most frequent 2×2, 2×3, 2×4, and 3×2 skipgram classes. Relative frequencies have been scaled by 10^3 and rounded appropriately.

8. REFERENCES

[1] D. Conklin. "Representation and Discovery of Vertical Patterns in Music". In: *Music and Artificial Intelligence*. Ed. by C. Anagnostopoulou, M. Ferrand, and A. Smaill. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 32–42. ISBN: 978-3-540-45722-0.

[2] D. Conklin and M. Bergeron. "Discovery of Contrapuntal Patterns". In: *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13, 2010*. Ed. by J. S. Downie and R. C. Veltkamp. International Society for Music Information Retrieval, 2010, pp. 201–206. ISBN: 978-90-393-5381-3.

[3] R. Gjerdingen. *Music in the Galant Style*. Oxford, New York: Oxford University Press, Oct. 11, 2007. 528 pp. ISBN: 978-0-19-531371-0.

[4] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks. "A Closer Look at Skip-Gram Modelling". In: *Proc. of the 5th International Conference on Language Resources and Evaluation (LREC-2006)*. European Language Ressources Association, 2006, pp. 1222–1225.

[5] D. Meredith, K. Lemström, and G. A. Wiggins. "Algorithms for Discovering Repeated Patterns in Multidimensional Representations of Polyphonic Music". In: *Journal of New Music Research* 31.4 (Dec. 1, 2002), pp. 321–345. ISSN: 0929-8215. DOI: 10.1076/jnmr.31.4.321.14162.

[6] M. T. Pearce and G. A. Wiggins. "Expectation in Melody: The Influence of Context and Learning". In: *Music Perception: An Interdisciplinary Journal* 23.5 (July 1, 2006), pp. 377–405. ISSN: 0730-7829, 1533-8312. DOI: 10.1525/mp.2006.23.5.377.

[7] M. Pearce and G. Wiggins. "Improved Methods for Statistical Modelling of Monophonic Music". In: *Journal of New Music Research* 33.4 (Dec. 1, 2004), pp. 367–385. ISSN: 0929-8215. DOI: 10.1080/0929821052000343840.

[8] D. R. W. Sears, A. Arzt, H. Frostel, R. Sonnleitner, and G. Widmer. "Modeling Harmony with Skip-Grams". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017*. Ed. by S. J. Cunningham, Z. Duan, X. Hu, and D. Turnbull. 2017, pp. 332–338. ISBN: 978-981-11-5179-8.

[9] R. P. Whorley, C. Rhodes, G. Wiggins, and M. T. Pearce. "Harmonising Melodies: Why Do We Add the Bass Line First?" In: International Conference on Computational Creativity. Sydney, 2013, pp. 79–86.

[10] R. P. Whorley, G. Wiggins, C. Rhodes, and M. T. Pearce. "Development of Techniques for the Computational Modelling of Harmony". In: International Conference on Computational Creativity. Coimbra, Portugal, 2010, pp. 11–15.