Structured Polyphonic Patterns

Mathieu Bergeron **Darrell Conklin**
Department of Computing
City University London
{bergeron,conklin}@soi.city.ac.uk

ABSTRACT

This paper presents a new approach to polyphonic music retrieval, based on a structured pattern representation. Polyphonic patterns are formed by joining and layering pattern components into sequences and simultaneities. Pattern components are conjunctions of features which encode event properties or relations with other events. Relations between events that overlap in time but are not simultaneous are supported, enabling patterns to express many of the temporal relations encountered in polyphonic music. The approach also provides a mechanism for defining new features. It is illustrated and evaluated by querying for three musicological patterns in a corpus of 185 chorale harmonizations by J.S.Bach.

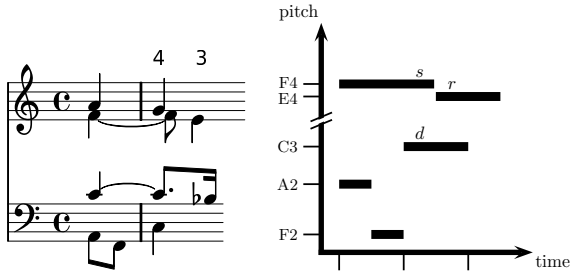

Figure 1. A 4-3 suspension between bass and alto voices in bars 16-17 of Bach's chorale BWV 283 (left) and a piano-roll representation of the alto and bass voices (right).

1 INTRODUCTION

1.1 Motivation

The concept of pattern in music is very important for diverse computational problems: database query and indexing; statistical modeling of musical style; and computer assisted musical analysis. Recently, there have been efforts to extend monophonic patterns to the general polyphonic case. Though there is agreement on the expressive requirements of monophonic patterns (e.g. transposition invariance), a consensus is yet to emerge in the polyphonic case. It is clear that patterns must be able to represent sequences of simultaneous events (simple first species counterpoint) and note against note counterpoint (second, third species). In addition, it is important that patterns represent relations between events that are overlapping but not simultaneous (fourth, fifth species).

Formalizing the knowledge found in treatises on counterpoint is a promising way to propose key features of polyphonic music, which can then be reused in other applications such as machine learning. This paper introduces structured polyphonic patterns ($\mathcal{SPP}$) and shows how they can be used to define polyphonic features and patterns for the musicological notions of parallel fifth, suspension, and cadential voice leading.

The concept of a *suspension* in counterpoint provides a useful case study for the expressive power required for polyphonic patterns. In a suspension (see Figure 1, left), a dissonance formed on a strong beat is resolved by step to a consonance. The pattern presented here is restricted to a simpler "proto-suspension" pattern for ease of exposition. The temporal relations and consonance and dissonance are captured, but features that refer to strong and weak beats, stepwise resolution, and relative duration of preparation and resolution notes are omitted.

Figure 1 (right) shows the notes involved in the suspension in piano-roll notation. The pattern involves three notes: the suspended note s; the note d introducing a dissonance; and the resolution note r (respectively F4, C3 and E4). A suspension pattern must precisely represent the temporal relations between these three notes. In particular, the pattern must represent that d starts while s is unfolding and that r starts while d is unfolding. At the same time, the pattern must represent harmonic pitch class intervals and also group these into classes of consonant and dissonant intervals. In addition, the suspension might occur between any two voices in a multiple voice texture, hence the pattern must capture voice combinations.

The Humdrum toolkit [5] supports polyphonic pattern representation and retrieval. However, the development of a polyphonic pattern can be a prohibitively complex task. To represent the suspension, the encoding of the source needs to be first transformed to a form where a single regular expression can test for the "starts while" temporal relation and

bass	tenor	alto	sop.
8AA	[4c	[4f	4a
8FF	.	.	.
=	=	=	=
4C	8.c]	8f]	4g
.	.	4e	.

bass	alto	hint	pc	cons
AA	f	-20	8	T
FF	(f)	-24	0	T
C	(f)	-17	5	F
(C)	e	-16	4	T

```
[a-g A-G]+[- # n]*[^)]*[(]   [a-g A-G]+[- # n]*.*F$
[a-g A-G]+[- # n]*[)]   [^(]*[a-g A-G]+[- # n]*.*T$
```

Figure 2. Hudrum/kern representation of the 4-3 suspension of Figure 1 before (top left) and after (top right) preprocessing. Simplified suspension pattern in Humdrum (bottom).

the occurrence of a consonance. Consider, for example, the last line of the top left of Figure 2. The E4 in the alto voice starts while the C3 in the bass voice is unfolding, introducing a consonant interval. The null token ".", used to indicate that C3 is still unfolding, must be replaced by the duplicated C3 (to allow harmonic interval computation), but also placed in round brackets to indicate that it is a continuation and does not represent the onset of a new event (this can be done with the Humdrum "ditto" command). Several additional preprocessing steps are necessary, for example: ties must be removed (otherwise the second component of a tie will not appear as an unfolding event); slurs must be removed (as slurs are also denoted by round brackets); and lines containing only null markers "." must be removed (otherwise, a suspension could span three lines instead of two).

Once preprocessing steps are executed, the musical content at the top left of Figure 2 is transformed into a form ready for querying, shown at the top right of Figure 2, which includes all additional columns needed in the pattern matching phase. These additional columns can be computed using Humdrum commands such as "hint", "pc", and "recode".

The pattern matching itself is executed by searching for two successive lines that respectively satisfy the regular expressions shown at bottom of Figure 2. Note that this will only find instances in which the d note appears in the first column (the bass voice in Figure 2). To capture every suspension in Humdrum, one must iterate the preprocessing steps, extracting every two voice combination, and testing for both ordering of those two voices. This must be programmed at a scripting language level above Humdrum.

2 METHODS

As the example above illustrated, the expression of even a simple polyphonic pattern is difficult in Humdrum. A central aim of this research is a pattern description and matching algorithm that is a more usable and efficient alternative to Humdrum. This section describes $\mathcal{SPP}$, a polyphonic pattern language inspired by algebraic representations of music [1, 4] and music knowledge representation methods [2].

2.1 Pattern components

Pattern components in $\mathcal{SPP}$ are represented using *feature sets*. A feature f has a *feature name* τ, a *feature value* v, and optionally a *voice* γ:

Definition 1 $\quad f \quad ::= \quad \tau : v \quad | \quad \tau(\gamma) : v$

A specific feature name is taken from a set of feature names (e.g. `pitch`). A feature value is either a value (e.g. 60) or a value variable. A voice is either a voice name or a voice variable. A feature of the form $\tau : v$ can be primitive (e.g. $\texttt{pitch} : 60$) or can encode a relation with an event occurring in the same voice (e.g. melodic interval $\texttt{mi} : -1$). A feature of the form $\tau(\gamma) : v$ is used to encode a relation with an event occurring in some other voice; for example, the feature $\texttt{vi}(\texttt{alto}) : -5$ encodes a vertical interval of -5 with some event in the alto voice.

A *pattern component* α has a feature set and a voice:

Definition 2 $\quad \alpha \quad ::= \quad \{f, \ldots, f\}_\gamma$

Example *E1* below illustrates a component containing two value variables (in upper case) and one voice variable (in lower case):

$$\{\texttt{duration} : D,\ \texttt{pitch} : P\}_y \qquad \textit{(E1)}$$

An *event* is a component with no variables, whose feature set uses at least the following feature names: `onset`, `duration` and `pitch` (omitted if the event is a rest). For example, the following event encodes the r note of Figure 1 (using MIDI pitch numbers and MIDI ticks at a resolution of 24 ticks per quarter note):

$$\{\texttt{onset} : 36,\ \texttt{duration} : 24,\ \texttt{pitch} : 64\}_{\texttt{alto}} \qquad \textit{(E2)}$$

A component α *matches* an event if, after assignments of the variables of α to values, the feature set of α is a subset of the feature set of the event and their voices are equal. For example, the component *E1* matches the event *E2* with the variable assignments $D \mapsto 24$, $P \mapsto 64$ and $y \mapsto \texttt{alto}$.

2.2 Pattern construction

Patterns in $\mathcal{SPP}$ are formed by joining components sequentially using the ";" operator and by layering components vertically using the "$=\!=$" operator. In addition, a component

may be modified by the "$-$" operator. Formally, the set of $\mathcal{SPP}$ patterns is defined inductively as follows:

Definition 3

$$\begin{array}{rcl} \phi & ::= & \alpha \\ & | & -\alpha \\ & | & \phi \,;\, \phi \\ & | & \dfrac{\phi}{\phi} \end{array}$$

To illustrate $\mathcal{SPP}$ semantics, the "proto-suspension" pattern will now be developed in multiple steps. First, the following pattern captures the sequence formed by the s and r notes in Figure 1:

$$\{\texttt{pitch} : 65\}_{\texttt{alto}} \;\; ; \;\; \{\texttt{pitch} : 64\}_{\texttt{alto}} \qquad \textit{(E3)}$$

This construction enforces a single temporal relation: the event matching the left component must directly precede the event matching the right component (i.e. the event matching the left component must end exactly as the event matching the right component starts). To capture both the alto and bass voice of Figure 1 (in particular to capture the d note of the suspension), two layers are joined as follows:

$$\frac{\{\texttt{pitch} : 65\}_{\texttt{alto}}}{\{\}_{\texttt{bass}}} \;\; ; \;\; \frac{\{\texttt{pitch} : 64\}_{\texttt{alto}}}{\{\texttt{pitch} : 48\}_{\texttt{bass}}} \qquad \textit{(E4)}$$

This construction enforces four temporal relations: i) the event matching the top left component must directly precede the event matching the top right component; ii) the event matching the bottom left component must directly precede the event matching the bottom right component; iii) the events matching the top and bottom right components must start together; and iv) the events matching the top and bottom left components must start together. Consequently, example *E4* does not correctly capture the temporal relations of a suspension, and the "$-$" operator is used to extend the pattern as follows:

$$\frac{-\{\texttt{pitch} : 65\}_{\texttt{alto}}}{-\{\}_{\texttt{bass}}} \;\; ; \;\; \frac{\{\texttt{pitch} : 64\}_{\texttt{alto}}}{-\{\texttt{pitch} : 48\}_{\texttt{bass}}} \qquad \textit{(E5)}$$

This construction also enforces four temporal relations. The first two are unchanged. As a modified component is layered with a non-modified component (*E5*, right), the third condition becomes: iii) the event matching the top right component must start while the event matching the bottom right component is unfolding. As two modified component are layered (*E5*, left), the fourth condition becomes: iv) the events matching the top and bottom left components must overlap. Note how the "$-$" does not represent a tie between the two bass components. The construction correctly captures the temporal relations between the s, d and r notes in the suspension example of Figure 1. Example *E5* can now be generalized to account for transposition invariance. This is done by replacing the concrete pitch features by more abstract vertical interval features:

$$\frac{\boxed{s}}{\dfrac{-\{\}_{\texttt{alto}}}{-\{\}_{\texttt{bass}}}} \;\; ; \;\; \frac{\boxed{r}}{\dfrac{\{\texttt{vi}(\texttt{bass}) : -16\}_{\texttt{alto}}}{-\{\texttt{vi}(\texttt{alto}) : 17\}_{\texttt{bass}}}} \qquad \textit{(E6)}$$

$$\boxed{d}$$

The feature $\texttt{vi}(\texttt{alto}) : 17$ specifies that the d note forms a vertical interval of 17 semitones with some overlapping note in the alto voice. This correctly represents the vertical interval between the d note and the s note. However, it lacks precision as it could also represent an interval between the d note and the r note. The feature $\texttt{vi}(\texttt{bass}) : -16$ also lacks precision as it specifies that the r note forms a vertical interval of -16 with any overlapping note in the bass voice. Therefore, the vertical interval feature must be specialized to the the "start while" temporal context:

$$\frac{-\{\}_{\texttt{alto}}}{-\{\}_{\texttt{bass}}} \;\; ; \;\; \frac{\{\texttt{sw\_vi}(\texttt{bass}) : -16\}_{\texttt{alto}}}{-\{\texttt{sw\_vi}(\texttt{alto}) : 17\}_{\texttt{bass}}} \qquad \textit{(E7)}$$

The feature $\texttt{sw\_vi}(\texttt{alto}) : 17$ is restricted to vertical intervals formed when the d note starts while some note in the alto voice unfolds. Therefore, it unambiguously represents the interval between the d note and the s note. Similarly, the feature $\texttt{sw\_vi}(\texttt{bass}) : -16$ unambiguously represents the vertical interval between the r note and the d note. Further generalization is achieved by replacing vertical intervals with classes of consonant and dissonant intervals:

$$\frac{-\{\}_{\texttt{alto}}}{-\{\}_{\texttt{bass}}} \;\; ; \;\; \frac{\{\texttt{sw\_cons}(\texttt{bass}) : \texttt{t}\}_{\texttt{alto}}}{-\{\texttt{sw\_dis}(\texttt{alto}) : \texttt{t}\}_{\texttt{bass}}} \qquad \textit{(E8)}$$

The `sw_cons`/`sw_dis` features are similar to the `sw_vi` feature, except that the absolute value of the vertical interval is tested for inclusion (respectively exclusion), modulo twelve, in the following set : $\{0, 3, 4, 7, 8, 9\}$. Finally, voice variables are used to capture suspensions between any two voices, resulting in the final "proto-suspension" pattern *P1*:

$$\frac{-\{\}_{x}}{-\{\}_{y}} \;\; ; \;\; \frac{\{\texttt{sw\_cons}(y) : \texttt{t}\}_{x}}{-\{\texttt{sw\_dis}(x) : \texttt{t}\}_{y}} \qquad \textit{(P1)}$$

In any instance, all three occurrences of the voice variable x (respectively y) must be assigned to the same voice name (i.e. the scope of variables is the whole pattern).

2.3 Defining polyphonic features

A strength of $\mathcal{SPP}$ is that the polyphonic features used in the previous section are given precise formulations in terms of feature definition rules. For example, the `sw_vi` feature is defined as follows:

$$\begin{array}{cc} \textbf{where} & \textbf{add} \\ \dfrac{\{\texttt{pitch}:P\}^*_x}{-\{\texttt{pitch}:Q\}_y} & \texttt{sw\_vi}(y): Q-P \end{array} \qquad (R1)$$

The **where** part uses a pattern to indicate that the `sw_vi` feature is formed wherever a note in voice x starts while a note in voice y is unfolding. The value variable P (respectively Q) is used to capture the pitch of the note in voice x (respectively y). The **add** part defines the feature to add. It is evaluated after the variables of the **where** part have been assigned. The resulting feature is added to the event matching the distinguished component in the **where** part (indicated with an asterisk in rule *R1*).

The same mechanism can also accommodate horizontal features, such as the melodic interval:

$$\begin{array}{cc} \textbf{where} & \textbf{add} \\ \{\texttt{pitch}:Q\}_x \,;\, \{\texttt{pitch}:P\}^*_x & \texttt{mi}: P-Q \end{array} \qquad (R2)$$

Applying the rules *R1* and *R2* to the source excerpt of Figure 1 would add the following features to the r note:

$$\begin{array}{l} \texttt{mi}: -1 \\ \texttt{sw\_vi}(\texttt{bass}): -16 \end{array} \qquad (E9)$$

Matching the **where** part of a feature definition rule is done with the same algorithm used for $\mathcal{SPP}$ pattern matching.

2.4 Pattern matching algorithm

The matching algorithm proceeds in two phases. In the first phase, the corpus is scanned and every component of the pattern is given an instance list (i.e. a list of matching events and corresponding variable assignments). In the second phase, the pattern is recursively analyzed and instance lists are joined, either according to a ";" operator or a "$==$" operator. Joins that do not respect $\mathcal{SPP}$ semantics or result in inconsistent variable assignments are simply discarded. The joining of two instance lists is done efficiently using data structures that take advantage of the fact that pairs of instances resulting in valid joins are always neighbors in the time dimension.

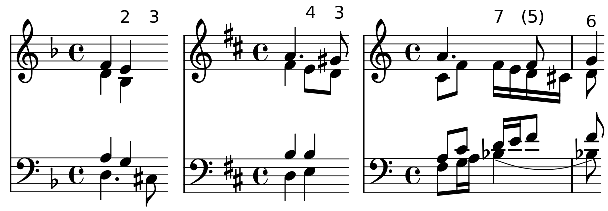

Figure 3. Examples of the two most frequent kinds of suspension found in the chorales: 2-3 suspension in BWV 272 bar 1 (left), and 4-3 suspension in BWV 262 bar 6 (middle). In addition, a 7-(5)-6 suspension in BWV 328 bars 32-33 (right).

3 RESULTS

An $\mathcal{SPP}$ parser and matching algorithm has been implemented in Ocaml and tested with 3 polyphonic patterns on a corpus of 185 chorale harmonizations by J.S.Bach. The corpus, originally encoded in the Humdrum format, was retrieved from www.kernscores.net [9] and saturated with the features described in Table 1.

3.1 Suspension

A total of 1177 instances of the suspension pattern *P1* developed in Section 2 were found in the corpus. Some instances of the suspension pattern are shown in Figure 3. The first two instances are examples of two of the four most frequent kinds of suspension found: 2-3 (509 instances), 4-3 (278 instances), 7-6 (128 instances) and 4-5 (98 instances). Note how the d note (the note introducing a dissonance) can be either in the upper voice (Figure 3, left) or lower voice (Figure 3, middle): components layered with the "$==$" operator may freely match any voice permutation.

An interesting instance of the pattern is illustrated at the right of Figure 3. In this example, a 7-6 suspension (A4-G4 in soprano) is delayed by an intermediate F4 in the soprano voice, forming a (consonant) interval of a fifth which matches the consonance captured by the `sw_cons` feature found in the top right component of *P1*. If desired, this type of match could be easily removed by introducing a feature representing stepwise motion in the same component.

3.2 Parallel fifth pattern

The second pattern selected for illustration is the parallel fifth, a voice leading structure avoided by Renaissance and tonal composers. In the corpus, 5 instances were discovered. Figure 4 presents two instances. The remaining instances can be found in BWV 263 bar 6, BWV 301 bar 3, and BWV 361 bar 12. The results here are identical (on a smaller data set) to those reported by Fitsioris and Conklin [3] who use

Feature name	Range	Description
pc	$\{0,\ldots,11\}$	Modulo twelve of pitch (pitch class)
pc_r	$\mathbb{B}$	True if the last pc is repeated
mi	$\mathbb{Z}$	Melodic interval (pitch difference with previous note)
mi_m	$\{0,\ldots,11\}$	Modulo twelve of mi (pitch class interval)
vi(x)	$\mathbb{Z}$	Vertical interval with an overlapping note in voice x
st_vi(x)	$\mathbb{Z}$	Vertical interval with a note in voice x that starts together with the current note
sw_vi(x)	$\mathbb{Z}$	Vertical interval with a note in voice x that is unfolding while the current note starts
et_vi(x)	$\mathbb{Z}$	Vertical interval with a note in voice x that ends together with the current note
vi_r(x)	$\mathbb{B}$	True if the last vertical interval with voice x is repeated
et_vi_am(x)	$\{0,\ldots,11\}$	Modulo twelve of the absolute value of et_vi
sw_cons(x)	$\mathbb{B}$	True if sw_vi(x) corresponds to a consonant interval
sw_dis(x)	$\mathbb{B}$	True if sw_vi(x) corresponds to a dissonant interval

Table 1. List of user-defined features used in this paper (top: horizontal features; bottom: vertical features). The ranges $\mathbb{Z}$ and $\mathbb{B}$ respectively refer to integers and booleans.

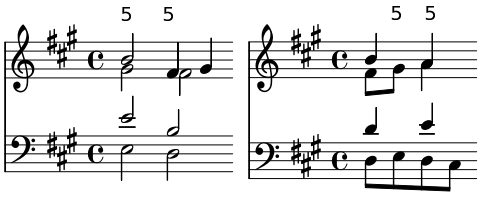

Figure 4. Selected instances of the parallel fifth pattern: BWV 323 bar 8 (left) and BWV 355 bar 15 (right).

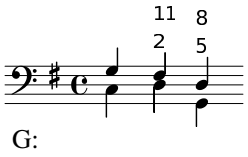

Figure 5. Instance of the cadential pattern in the bass and tenor voices: BWV 293 bar 8

a Prolog encoding of the parallel fifth pattern and provide a musicological interpretation of all instances.

The $\mathcal{SPP}$ pattern of the parallel fifth was carefully designed to avoid false positives (e.g. antiparallel fifths, cases separated by rests):

$$\{\texttt{et\_vi\_am}(y):7\}_x \;;\; \{\texttt{vi\_r}(y):\texttt{t},\ \texttt{pc\_r}:\texttt{f}\}_x \qquad \textit{(P2)}$$

The first component of *P2* captures a vertical interval of a perfect fifth (7 semitones modulo twelve, hence compound fifths are also captured). The second component ensures that the fifth is repeated (vi_r(y) : t feature) and that there is melodic motion (pc_r : f feature). The vi_r(y) : t feature uses a voice variable to ensure that the successive fifths occur between the same two voices (the current voice x and some other voice y).

3.3 Cadential melodic interval pattern

The final pattern reported in this paper was initially reported in the context of the discovery of vertical patterns in the Bach chorales [1]. It was observed that a particular structure of two simultaneous pitch class intervals (2-5 in the bass and 11-8 in a higher voice; Figure 5) occurs at many cadences. Interestingly, in these cadences the leading tone falls to the fifth scale degree rather than rising to the tonic. Here, a two-voice fragment of that pattern is encoded as follows:

$$\frac{\{\}_x}{\{\}_{\texttt{bass}}} \;;\; \frac{\{\texttt{mi\_m}:11\}_x}{\{\texttt{mi\_m}:2\}_{\texttt{bass}}} \;;\; \frac{\{\texttt{mi\_m}:8\}_x}{\{\texttt{mi\_m}:5\}_{\texttt{bass}}} \qquad \textit{(P3)}$$

The query on the corpus returned a total of 70 instances. In most (64) of the instances, the first chord is a ii^6_5. The first instance of Figure 6 illustrates such a case. It contains the pattern in the bass and alto lines, with extensive melodic and rhythmic elaboration occurring in the surrounding tenor and soprano voices. In the remaining 6 cases a IV or IV^7 is formed on the first chord. Figure 6 (right) illustrates such a IV^7 case. Note that these latter cases could easily be excluded by adding the feature st_vi(y) : 9 to the component at the bottom left of *P3*. This would capture the vertical interval of a major sixth (9 semitones) between the bass note and some note in another voice, which has to occur in a ii^6_5 chord.

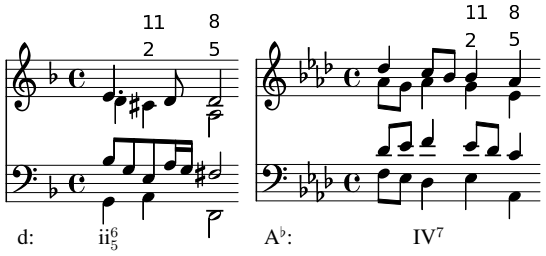

Figure 6. Selected instances of the two-voice cadential pattern: BWV 297 bar 11 (left) and BWV 354 bar 6 (right).

4 DISCUSSION

The paper introduced $\mathcal{SPP}$, a structured polyphonic pattern language and matching algorithm. An important feature of $\mathcal{SPP}$ is that voicing is handled in a general way and that voice permutations are explored automatically. It also provides a simple feature definition mechanism to offer a great deal of flexibility.

Most existing approaches to polyphonic pattern representation lack the expressiveness to accurately capture the patterns discussed in this paper. For example, vertical patterns [1] can only match polyphonic sources that have been expanded and sliced to yield a homophonic texture. A point set pattern representation [7, 8] can only encode a class of suspensions for which the duration ratios and the vertical intervals are always the same (capturing every suspension would require a set of patterns, the size of which can grow quickly as many different ratios and vertical intervals are likely to be found in the source). Techniques that rely on approximate matching to a source fragment [6] can confuse simultaneous notes with notes that overlap without being simultaneous. This can result in a significant loss of precision when retrieving patterns such as the suspension in which events overlap but are not simultaneous. In comparison with Humdrum, $\mathcal{SPP}$ shifts the kind of understanding required from operational (understanding the processing steps required to search for a pattern) to denotational (understanding the meaning of particular features and patterns). This shift offers many advantages, such as the ease to specialize a pattern, and a greater confidence in the precision of the query. Finally, $\mathcal{SPP}$ is much easier to extend for a researcher, as adding a new operator can be done without significantly modifying the pattern matching algorithm.

In the future, the formal properties of the language will be explored, e.g. the possibility of reasoning about pattern equivalence. Also, the expressiveness of $\mathcal{SPP}$ will be formally investigated (including a comparison with Humdrum). Further evaluation of the approach will be achieved by encoding more patterns (e.g. appoggiatura, chained suspensions, neighbor tones and passing tones, cross relation) and analyzing the result of matching in wider more diverse corpora, including corpora of piano music where voices can appear or disappear (this paper used a corpus that had an unchanging four-voice texture). The resulting catalog of polyphonic patterns might be used as global piece features for machine learning. Finally, the topic of pattern discovery based on the $\mathcal{SPP}$ language will be explored.

5 REFERENCES

[1] D. Conklin. Representation and discovery of vertical patterns in music. In C. Anagnostopoulou, M. Ferrand, and A. Smaill, editors, *Music and Artificial Intelligence: Lecture Notes in Artificial Intelligence*, number 2445, pages 32–42. Springer-Verlag, 2002.

[2] D. Conklin and M. Bergeron. Representation and discovery of feature set patterns in music. *Computer Music Journal*, 32(1):60–70, 2008.

[3] G. Fitsioris and D. Conklin. Parallel successions of perfect fifths in the Bach chorales. In *Fourth Conference on Interdisciplinary Musicology*. Thessaloniki, Greece, 2008.

[4] P. Hudak, T. Makucevich, S. Gadde, and B. Whong. Haskore music notation - an algebra of music. *Journal of Functional Programming*, 6(3):465–483, 1996.

[5] D. Huron. Music research using Humdrum: A user's guide. Stanford, California: Center for Computer Assisted Research in the Humanities, 414 pages. 1999.

[6] S. T. Madsen and G. Widmer. Evolutionary search for musical parallelism. In *Applications of Evolutionary Computing, proceedings of the EvoWorkshops 2005*, Lecture Notes in Computer Science, pages 488–497. Springer Verlag, 2005. Lausanne, Switzerland, 2005.

[7] D. Meredith, K. Lemström, and G. A. Wiggins. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*, 31(4):321–345, 2002.

[8] C. A. Romming and E. Selfridge-Field. Algorithms for polyphonic music retrieval: the Hausdorff metric and geometric hashing. In *International Conference on Music Information Retrieval (ISMIR)*, pages 457–462. Vienna, Austria. 2007.

[9] C. Sapp. Online database of scores in the Humdrum file format. In *International Conference on Music Information Retrieval (ISMIR)*, pages 664–665. London, United Kingdom. 2005.