LEARNING AND VISUALIZING MUSIC SPECIFICATIONS USING PATTERN GRAPHS

Rafael Valle[1] **Daniel J. Fremont**[2] **Ilge Akkaya**[2]
Alexandre Donze[2] **Adrian Freed**[1] **Sanjit S. Seshia**[2]
[1] UC Berkeley, CNMAT
[2] UC Berkeley
rafaelvalle@berkeley.edu

ABSTRACT

We describe a system to learn and visualize specifications from song(s) in symbolic and audio formats. The core of our approach is based on a software engineering procedure called specification mining. Our procedure extracts patterns from feature vectors and uses them to build pattern graphs. The feature vectors are created by segmenting song(s) and extracting time and and frequency domain features from them, such as chromagrams, chord degree and interval classification. The pattern graphs built on these feature vectors provide the likelihood of a pattern between nodes, as well as start and ending nodes. The pattern graphs learned from a song(s) describe formal specifications that can be used for human interpretable quantitatively and qualitatively song comparison or to perform supervisory control in machine improvisation. We offer results in song summarization, song and style validation and machine improvisation with formal specifications.

1. INTRODUCTION AND RELATED WORK

In software engineering literature, specification mining is an efficient procedure to automatically infer, from empirical data, general rules that describe the interactions of a program with an application programming interface (API) or abstract datatype (ADT) [3]. It has convenient properties that facilitate and optimize the process of developing formal specifications. Specification mining is a procedure that is either entirely automatic, or only requires the relatively simple task of creating templates. It offers valuable information on commonalities in large datasets and exploits latent properties that are unknown to the user but reflected in the data. Techniques to automatically generate specifications date back to the early seventies, including [5, 24]. More recent research on specification mining includes [2,3,10,17]. In general, specification mining tools mine temporal properties in the form of mathematical logic or automata. Figure 1 describes a simple musical specification. Broadly speaking, the two main strategies for building these automata include: 1) learning a single automaton and inferring specifications from it; 2) learning small templates and designing a complex automaton from them. For example, [3] learns a single probabilistic finite state automaton from a trace and then extracts likely properties from it. The other strategy circumvents the NP-hard challenge of directly learning a single automaton [14, 15] by first learning small specifications and then post-processing them to build more complex state machines. The idea of mining simple alternating patterns was introduced by [10], and subsequent efforts include [12, 13, 25, 26].

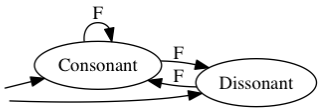


Figure 1: This graph describes three specifications: 1) a sequence must start (unlabelled incoming arrow) with a note of any type; 2) every note that does not belong to the underlying chord (dissonant) must be followed by a note that belongs to that chord (consonant); 3) a consonant note must be followed by a dissonant note or another consonant note; F means followed.

Manually describing such general rules from music is a complex problem, even for experts, due to music's parameter space complexity and richness of interpretation. Specification mining is a very attractive solution because it offers a systematic and automatic mechanism for learning these specifications from large amounts of data. Similar to specification mining strategies, algorithms for pattern discovery in music such as [6, 20, 21] combine segmentation and exhaustive search to find patterns that will be condensed to create a statisticailly significant description of the song(s). Our method avoids the exhaustive search by searching for specific patterns and creates a complex pattern graph by combining the patterns found, combining pattern graphs, and recursively building pattern graphs learned from pattern graphs. The pattern graph allows the representation of edges and nodes as mathematical objects, e.g. multidimensional point sents or Gaussian Mixture Models (GMM), hence it is not limited to strings.

2. SPECIFICATIONS AND PATTERN GRAPH

This paper adapts the work of [17] to formally describe specification mining in music. It expands our previous efforts in [9] by developing an inference engine that uses pre-defined templates to mine from a collection of traces (songs) specifications in the form of pattern graphs.

2.1 Formal Definition

Let F be a list of features extracted from a song $\mathcal{S}$, e.g. pitch, duration, chroma, etc. The notation $v_{f,t}$ indicates the value of $f \in F$ at time t.

Definition 1 (Event) *Formally, we define an event with the tuple $(\vec{f}, \vec{v}, t)$, where $\vec{f}$ is a set of features and $\vec{v}$ is their corresponding values at time t. The alphabet Σ_f is the set of distinct events given feature f, and a finite trace τ is a sequence of events ordered by their time of occurrence.*

Definition 2 (Projection) *The projection π of a trace τ onto an alphabet Σ, $\pi_\Sigma(\tau)$, is defined as τ with all events not in Σ deleted.*

Definition 3 (Specification Pattern) *A specification pattern is a finite state automata, FSA, over symbols Σ. Patterns can be parametrized by the events used in this alphabet; for example, we use "the $\mathcal{A}$ pattern between events a and b" to indicate the pattern obtained by taking a FSA $\mathcal{A}$ with $|\Sigma| = 2$ and using a as the first element of Σ and b as the second. A pattern is satisfied over a trace τ with alphabet $\Sigma_\tau \supseteq \Sigma$ iff $\pi_\Sigma(\tau) \in \mathcal{L}(\mathcal{A})$, that is, if and only if the projection of the trace onto the alphabet Σ is in the language of A.*

Definition 4 (Binary Pattern) *A binary pattern is a specification pattern with alphabet size 2. We denote a binary pattern between events a and b as a **R** b, where **R** is a label identifying the pattern.* [1]

Definition 5 (Pattern Graph) *A pattern graph is a labelled directed multigraph whose nodes are elements of Σ_f , i.e. values of a feature f . A node can be labelled as a starting node, an ending node, or neither. Edges are labelled with a type of binary pattern and a count indicating how many times the pattern occurred in the dataset used to build the pattern graph.*

For example, an edge (a, b) labelled (R, 3) in the pattern graph means the pattern a **R** b occurred 3 times in the dataset. Figure 3 provides a complete example of a pattern graph learned from the example in Figure 2. We have indicated starting nodes with an unlabelled incoming arrow and ending nodes with a double circle (by analogy to the standard notation for FSAs). [2]

[1] Although we explored binary patterns in this paper, our method supports patterns with more than two events.

[2] A pattern graph can be converted into an automaton, but is not itself an automaton.

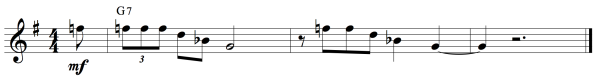


Figure 2: First phrase of Crossroads Blues by Robert Johnson as transcribed in the Real Book of Blues. The transition from chord degree 10 (note f) to chord degree 7 (note d) is always preceded by two or several occurrences of chord degree 10. Not merging *10* **F** *7* with *10* **F** *7* represents a musical inconsistency and the pattern graph would accept words such as $(10, 7, 10, 7)$.

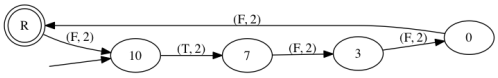


Figure 3: Pattern graph learned on the chord degree feature (interval from root) extracted from the phrase in Fig. 2. The **F** pattern between chord degrees 10 and 7 has been merged into the pattern 10 **T** 7.

2.2 Patterns

We generate specifications by mining small patterns from a set of traces and combining the mined patterns into a pattern graph. The patterns in this paper as described as regular expressions, *re*, and were chosen based on idiomatic music patterns such as repetition and ornamentation. Other patterns can be mined by simply writing their *re*.

Followed(F): This pattern occurs when event a is immediately followed by event b. It provides information about immediate transitions between events, e. g. resolution of non-chord tones. We denote the followed pattern as *a* **F** *b* and describe it with the *re* (ab).

'Til(T): This pattern occurs when event a appears two or more times in sequence and is immediately followed by event b. It provides information about what transitions are possible after self-transitions are taken. We denote the 'til pattern as *a* **T** *b* and describe it with the *re* (aaa^*b).

Surrounding(S): This pattern occurs when event a immediately precedes and succeeds event b. It provides information over a time-window of three events and we musically describe it as an ornamented self-transition. We denote the surrounding pattern as *a* **S** *b* and describe it with the *re* (aba).

2.3 Pattern Merging

If every match to a pattern $P_2 = a\ \mathbf{R}\ b$ occurs inside a match to a pattern $P_1 = a\ \mathbf{Q}\ b$, we say that P_1 *subsumes* P_2 and write $P_1 \implies P_2$. When this happens, we only add the *stronger* pattern P_1 to the pattern graph, with the purpose of emphasizing longer musical structures. Given the patterns described in this paper:

1. *a* **T** *b* $\implies$ *a* **F** *a*, *a* **F** *a* is merged into *a* **T** *b*
2. *a* **T** *b* $\implies$ *a* **F** *b*, *a* **F** *b* is merged into *a* **T** *b*
3. *a* **S** *b* $\implies$ *a* **F** *b*, *a* **F** *b* is merged into *a* **S** *b*
4. *a* **S** *b* $\implies$ *b* **F** *a*, *b* **F** *a* is merged into *a* **S** *b*

Shorter patterns not included will be added *iff* they occur outside the scope of longer patterns. Nonetheless, the pattern graph is designed such that it accepts traces that satisfy the longer pattern, e.g. a **T** b accepts the sequences aab and $aaab$, but not ab or aac.

3. LEARNING AND ENFORCING SPECIFICATIONS

3.1 Learning Specifications

Given a song dataset and their respective features, we build pattern graphs $\mathcal{G}_f \in \mathcal{G}$ by mining the patterns described in 2. The patterns in $\mathcal{G}$ correspond to the set of allowed patterns, while all others are forbidden.

The synchronous product of $\mathcal{G}_f$ can be used to build a specification graph $\mathcal{G}^s$ that can be used to supervise the output of a machine improviser. This concept originates from the Control Improvisation framework, which we first introduced in [8, 9] and have used in IoT applications [1]. We refer the reader to [11] for a thorough explanation.

Algorithm 1 describes the specification mining algorithm. $\mathfrak{D}$ is a dataset, e.g. Table 1, containing time and frequency domain features, described in section 4, extracted from songs with or without phrase boundary annotations; $\mathcal{P}$ is a list containing string representations of the regular expressions that are used to mine patterns. The pattern graph implementation and the code used to generated this paper can be found on our github repository [3]

Algorithm 1: Specification Mining Algorithm

Input: dataset $\mathfrak{D}$ over features $\mathcal{F}$; patterns $\mathcal{P}$
Output: a pattern graph $\mathcal{G}_f$ for each $f \in \mathcal{F}$

```
for f ∈ F do
    G_f ← new pattern graph on vertices Σ_f
    for song ∈ D do
        for phrase ∈ song do
            phrase_f ← the sequence of values of the
            feature f in phrase
            label the first element of phrase_f as a
            starting node in G_f
            label the last element of phrase_f as an
            ending node in G_f
            for a, b ∈ Σ_f do
                counts ←
                countPatternMatches(a, b, phrase_f, P)
                foreach pattern P with
                counts(P) > 0 do
                    add to G_f the edge (a, b) with
                    label (P, counts(P))
```

In the next section we describe some of the features, or viewpoints, that we used in this paper to build specifications that describe relevant musical properties of a song(s).

[3] https://github.com/rafaelvalle/music_pattern_graphs

4. MUSIC SPECIFICATION MINING

We abstract and formalize a *song* into a sequence of feature values possibly aligned with a *chord progression*, phrase-segmented and including key signature changes. In this paper, the time unit is the *beat*, including respective integer subdivisions. To encode all events in a score, we use an alphabet which is the product of five alphabets: $\Sigma = \Sigma_p \times \Sigma_d \times \Sigma_a \times \Sigma_b \times \Sigma_{12}$, where

- Σ_p is the *pitches* alphabet, i.e. $\Sigma_p = \{\text{𝄽}, \texttt{a0}, \texttt{a\#0}, \cdots\}$;
- Σ_d is the *durations* alphabet, i.e. $\Sigma_d = \{\text{𝅘𝅥𝅯}, \text{♩}, \text{𝅗𝅥}, \ldots\}$ with $\text{♩} = 1$ beat. Note that Σ_d also includes positive integer subdivisions of the beat, e.g. for tuplets.
- Σ_c is the *chords* alphabet, i.e. $\Sigma_c = \{\text{C, D7\#4}, \ldots\}$;
- Σ_b is the *beat* alphabet. For example, if the smallest duration (excluding fractional durations) is the eighth and the meter is in 4, then Σ_b is $\{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5\}$.
- Σ_{12} is the binary *chroma* alphabet. For this, we interpret the binary chroma as a binary number and encode it with the respective Unicode string.

Note that the full alphabet enables the creation of data abstractions, e.g. chord degree. Below we describe the data abstractions implemented using the alphabet above. A similar strategy is used in [6, 7], where data abstractions (derived types, viewpoints) are implemented. In our current implementation, all the specifications implicitly use the full alphabet Σ via the product of pattern graphs.

4.1 Time Domain Features

- **Event Duration**: This feature describes the duration in beats of silences and notes. It imposes hard constraints on duration diversity but provides weak guarantees on rhythmic complexity because it has no awareness of beat location. Figure 4 provides one example of such weak guarantees. Further constraints can be imposed by combining event duration and beat onset location.

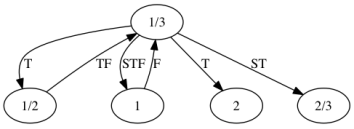


Figure 4: Selection of event duration specifications learned from a blues songs dataset. The pattern 1/3 **S** 1 (1/3, 1, 1/3) is allowed but can produce incomplete tuplets.

- **Beat onset location**: This feature describes where events happen within the beat. Cooperatively, event duration and beat onset location produce complex specifications that allow for rhythmic diversity. These specifications extend the work in [9] by replacing handmade specifications designed to ensure rhythmic tuplet completeness with specifications learned from data. Figure 5 provides an example of such specifications learned from 4/4 songs.

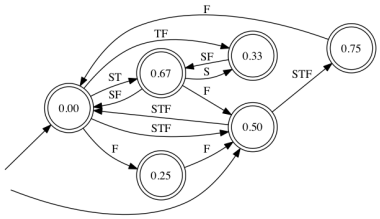


Figure 5: Beat onset location specifications learned from a blues songs dataset.

4.2 Frequency Domain Features

- **Scale Degree**: The scale degree is the identification of a note disregarding its octave but regarding its distance from a reference tonality. Songs usually impose soft constraints on the pitch space, defining the set of appropriate scale degrees and transitions thereof. Figure 6 provides a selection of scale degree mined specifications. Since scale degree can only provide overall harmonic constraints to each tone over the scope of the entire song, we use another feature to provide harmonic constraints based on chord progression, therefore increasing the temporal granularity of the harmonic specifications.

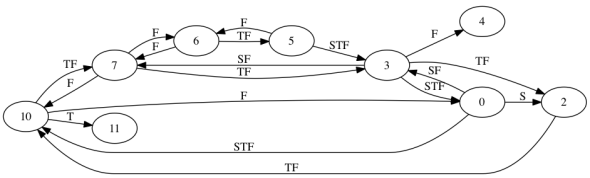


Figure 6: Selection of scale degree specifications learned from a blues songs dataset. These specifications conform with the general consent that blues songs include the main key's major scale with the "flat seven" (scale degree 10) and the blue note (scale degree 3). Note that sharp fourths (scale degree 6) are used as approach tones to scale degrees 5 and 6.

- **Interval Classification**: Expanding on [9], we replace the hand-designed tone classification specifications, here called interval classification, with mined specifications. These specifications provide information about the size (diatonic or leap) and quality (consonant or dissonant) of the music interval that precedes each tone. Figure 7 illustrates mined specifications. Although scale degree and interval classification specifications ensure desirable harmonic guarantees given key and chord progression, they provide no melodic contour guarantees.

- **Melodic Interval**: This feature operates on the first difference of pitch values and is associated with the contour of a melody. Combined with scale degree and interval classification, it provides harmonic and melodic constraints, including melodic contour.

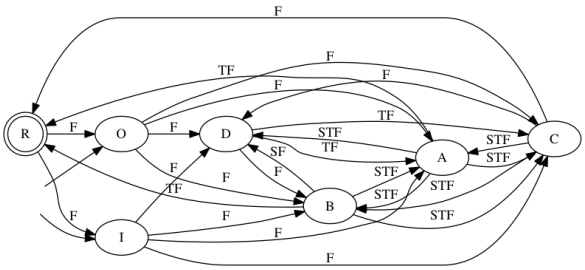


Figure 7: Interval class specifications learned from a blues songs dataset. The symbols A, B, C, and D, describe tones reached by consonant step, consonant leap, dissonant (non-chord tones) step, and dissonant leap respectively. Consonant and dissonant notes preceded by rests, R, are described with the symbols I and O respectively.

- **Chord Degree**: The chord degree is the identification of a note regarding its distance in semitones to the root of a chord. It adds harmonic specificity to the interval class.

Table 1 provides the reader with a selection of features extracted from a blues song with chord and phrase number annotations. The next section analyzes in detail the application of pattern graphs and specifications in song summarization, song and style validation, and machine improvisation with formal specifications.

5. EXPERIMENTAL RESULTS

For the experiments in this paper, we learned pattern graphs and pattern sequences from three non-overlapping datasets, namely:

$\mathfrak{D}^{train}$ a dataset of 20 blues songs with chord and phrase annotations, transcoded from the Real Book of Blues [18];

$\mathfrak{D}^{test}$ a dataset of 10 blues songs with chord and phrase annotations, transcoded from the Country Blues songbook [16];

SAC a dataset, with 10 genres and 25 pieces of music per genre [19].

pretty_midi [22] is used for handling midi data.

5.1 Style and Song Summarization

Pattern graph plots can be used to understand and visualize the patterns of a song or musical style. In section 4 we provided pattern graph visualizations that described significant musical properties of $\mathfrak{D}^{train}$. Pattern sequence plots, on the other hand, offer a visualization that is directly related to a song's formal structure. A pattern sequence plot is a color sequence visualization of a pattern sequence extracted from a song; for example, the chroma pattern sequence (100010010000, T, 000000000000, F, 100000000000), describes: play any inversion of the C

	chord	dur	measure	phrase	...	pitch	mel_interval	beat	interval_class
0	F7	14/3	1	1	...	69	R	1	I
1	F7	1/3	2	1	...	65	-4	5/3	B
2	F7	2/3	2	1	...	67	2	1	C
...	...	...	...	...	...	...	...	.....	
23	B-7	1	10	3	...	67	-1	1	C
24	F7	4	11	3	...	65	-2	1	A
25	F7	-4	12	3	...	R	R	1	R

Table 1: Dataframe from Blues Stay Away From Me by Wayne Raney et al. R represents a rest.

major triad two or more times, followed by one rest followed by the note C played one time. The conversion of a feature into color is achieved by mapping each feature dimension to RGB. Features with more than three dimensions undergo dimensionality reduction to a 3 dimensional space through non-negative matrix factorization (NMF) [4] . Figure 8 shows a plot of binary chroma and dimensionality reduced binary chroma overlaid with the patterns associated with each time step.

5.2 Song and Style Validation

Song and style validation describe to what extent a song or a style violates a specification. A violation occurs when a pattern does not satisfy a specification, i.e. the pattern does not exist in the pattern graph. Figure 9 provides histograms of violations obtained by validating $\mathfrak{D}^{test}$ on chord degree and interval specifications learned from $\mathfrak{D}^{train}$.

Given a total of 355 patterns learned from $\mathfrak{D}^{test}$, there were 35 chord degree violations and 35 melodic interval violations, producing an average violation ratio of ≈ 0.02 per song [5] . The dataset used for learning the specifications is small. A larger dataset will enable us to better investigate how they are not characteristic of the blues.

For the task of style validation, we build binary chroma specifications for each genre in the SAC dataset. The specifications are used separately to validate all genres in the SAC dataset. Validation is performed with the average validation ratio, which is computed as the ratio of violations given number of patterns in the song being validated. Figure 10 provides the respective violation ratio matrix. [6] These validations can be exploited in style recognition and we foresee that more complex validations are possible by using probabilistic metrics and describing pattern graph nodes as GMMs.

5.3 Machine Improvisation with formal specifications

Machine improvisation with formal specifications is based on the framework of Control Improvisation. Musically speaking, it describes a framework in which a *controller* regulates the events generated by an *improviser* such that all events generated by the *improviser* satisfy hard (non-probabilistic) and soft (probabilistic) specifications.

Using a 12-bar blues excerpt and its chord progression shown in Figure 12, we *navigated* the factor oracle [4] with 75% replication probability to generate improvisations with specifications generated from $\mathfrak{D}^{train}$. In this task we used duration, beat onset location, chord degree, interval class and melodic interval joint specifications.

We computed the average melodic similarity between $\mathfrak{D}^{train}$ and other sets of improvisation, including: 50 factor oracle improvisations generated without specifications, 50 factor oracle improvisations generated with specifications. The melodic similarity is computed using the algorithm described in [23]. As baselines, we also computed the similarity of $\mathfrak{D}^{train}$ to the 12 Bar Blues reference word and to 50 random improvisations. The results in Figure 11 show that the specifications are successful in controlling the events generated by the improviser, factor oracle, such that they are more similar to $\mathfrak{D}^{train}$ and satisfy the specifications learned from it.

Qualitatively, the improvisation without specifications violates several specifications related to expected harmonic and melodic behavior, as Figure 12 confirms. For example, measure 4 in the improvisation without specifications has chord degrees that violate harmonic specifications. This is possible because the events generated by the unsupervised improvisation disregard harmonic context, thus commonly producing unprepared and uncommon dissonant notes.

The improvisations with specifications are able to keep overall harmonic coherence despite the use of chromaticism. Their melodic contour is rather smooth and the improvisations include several occurrences of the 'Til and Surrounding patterns, as measures 5 and 1 respectively show.

6. CONCLUSIONS AND FUTURE WORK

This paper investigated the use of pattern graphs and specification mining for song and style summarization, validation, and machine improvisation with formal specifications. Our experimental results show that pattern graphs can be successfully used to graphically and algorithmically describe and compare characteristics of a music collection, and in guiding improvisations.

We are currently investigating smoothing strategies, including the use of a larger dataset, for pattern graph learning so that we can more robustly use probabilistic metrics for song and style validation.

[4] We use scikit-learn's NMF with default parameters

[5] $(35 + 35)/(355 * 10)$

[6] Note that this is not a confusion matrix and must not be symmetric.

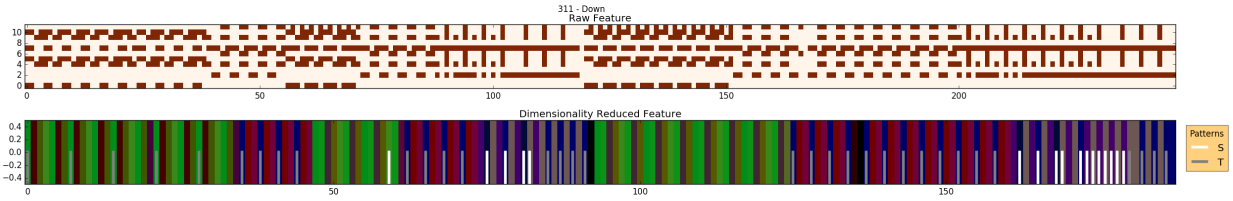


Figure 8: Down by the band 311 as found in SAC's dataset. The top plot shows the raw feature (binary chroma). The bottom plot shows the dimensionality reduced chroma (NMF with 3 components) with the components scaled and mapped to RGB. The patterns associated with each event are plotted as grayscale bars. The absence of a grayscale bar represents the Followed pattern.

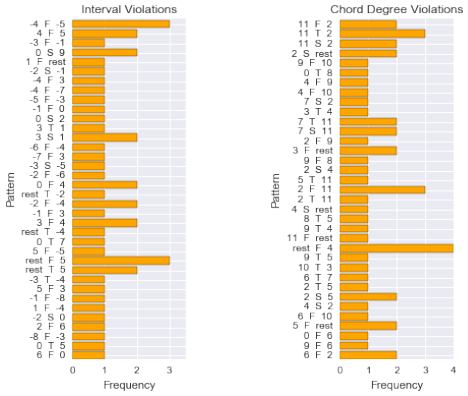


Figure 9: Histogram of melodic interval and chord degree violations. The y-axis represents the patterns that do not exist in the specification and the x-axis represents their frequency. F and T represent the patterns Followed and 'Till respectively.

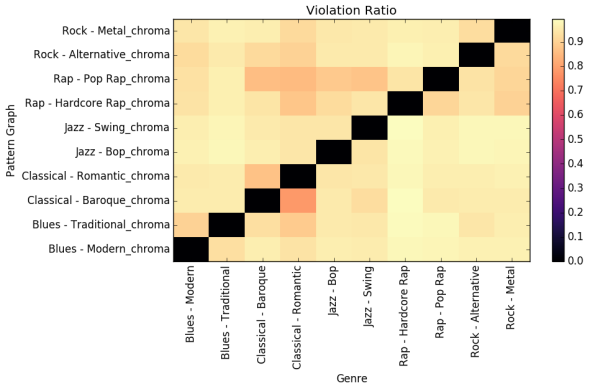


Figure 10: This violation ratio matrix shows that similar styles have lower violations ratios. Unexpectedly, Rap - Pop Rap specifications are more violated by Rock - Alternative than by Classical - Baroque or Classical - Romantic.

Although for this paper we hard-coded the pattern mining algorithm to avoid regular expression's long run time, we are researching sequential pattern mining algorithms that are fast and easy and flexible to use as *re*.

Last and most important, we are expanding specification mining to real-valued multidimensional features by expressing pattern graphs nodes as gaussian mixtures.

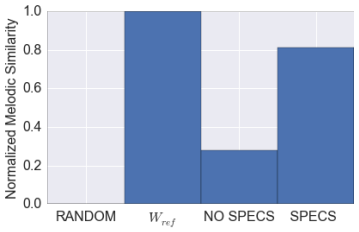
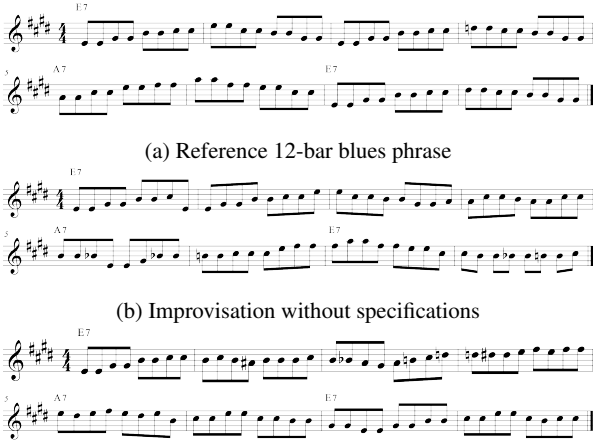


Figure 11: Normalized Melodic Similarity w.r.t $\mathfrak{D}^{train}$. W_{ref} is the 12-bar blues phrase used as improvisation input. NO SPECS and SPECS are improvisations generated with the factor oracle with 0.75 replication probability with and without specifications. The results show that specifications induce improvisations from that factor oracle that are closer to $\mathfrak{D}^{train}$.



(a) Reference 12-bar blues phrase

(b) Improvisation without specifications

(c) Improvisation with specifications

Figure 12: Factor Oracle improvisations with 0.75 replication probability on a traditional instrumental blues phrase.

7. ACKNOWLEDGEMENTS

This research was supported in part by the TerraSwarm Research Center, one of six centers supported by the STAR net phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

8. REFERENCES

[1] Ilge Akkaya, Daniel J. Fremont, Rafael Valle, Alexandre Donze, Edward A. Lee, and Sanjit A. Seshia. Control improvisation with probabilistic temporal specifications. In *IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI'16)*, 2015.

[2] Rajeev Alur, Pavol Cernỳ, Parthasarathy Madhusudan, and Wonhong Nam. Synthesis of interface specifications for java classes. *ACM SIGPLAN Notices*, 40(1):98–109, 2005.

[3] Glenn Ammons, Rastislav Bodík, and James R Larus. Mining specifications. *ACM Sigplan Notices*, 37(1):4–16, 2002.

[4] Gérard Assayag and Shlomo Dubnov. Using factor oracles for machine improvisation. *Soft Comput.*, 8(9):604–610, 2004.

[5] Michel Caplain. Finding invariant assertions for proving programs. In *ACM SIGPLAN Notices*, volume 10, pages 165–171. ACM, ACM, 1975.

[6] Darrell Conklin and Mathieu Bergeron. Feature set patterns in music. *Computer Music Journal*, 32(1):60–70, 2008.

[7] Darrell Conklin and Ian H Witten. Multiple viewpoint systems for music prediction. *Journal of New Music Research*, 24(1):51–73, 1995.

[8] Alexandre Donzé, Sophie Libkind, Sanjit A. Seshia, and David Wessel. Control improvisation with application to music. Technical Report UCB/EECS-2013-183, EECS Department, University of California, Berkeley, Nov 2013.

[9] Alexandre Donzé, Rafael Valle, Ilge Akkaya, Sophie Libkind, Sanjit A. Seshia, and David Wessel. Machine improvisation with formal specifications. In *Proceedings of the 40th International Computer Music Conference (ICMC)*, 2014.

[10] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. *Bugs as deviant behavior: A general approach to inferring errors in systems code*, volume 35. ACM, 2001.

[11] Daniel J. Fremont, Alexandre Donzé, Sanjit A. Seshia, and David Wessel. Control improvisation. *CoRR*, abs/1411.0698, 2014.

[12] Mark Gabel and Zhendong Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 339–349. ACM, 2008.

[13] Mark Gabel and Zhendong Su. Symbolic mining of temporal specifications. In *Proceedings of the 30th international conference on Software engineering*, pages 51–60. ACM, 2008.

[14] E. Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[15] E. Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

[16] Stefan Grossman, Stephen Calt, and Hal Grossman. *Country Blues Songbook*. Oak, 1973.

[17] Wenchao Li, Alessandro Forin, and Sanjit A. Seshia. Scalable specification mining for verification and diagnosis. In *Proceedings of the 47th design automation conference*, pages 755–760. ACM, 2010.

[18] Jack Long. The real book of blues. Wise, Hal Leonard, 1999.

[19] Cory McKay and Ichiro Fujinaga. Combining features extracted from audio, symbolic and cultural sources. In *ISMIR*, pages 597–602. Citeseer, 2008.

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

[21] Marcus Pearce. *The Construction and Evaluation of Statistical Models of Melodic Structure in Music Perception and Composition*. PhD thesis, School of Informatics, City University, London, 2005.

[22] Colin Raffel and Daniel PW Ellis. Intuitive analysis, creation and manipulation of midi data with pretty_midi. In *15th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers*, 2014.

[23] Rafael Valle and Adrian Freed. Symbolic music similarity using neuronal periodicity and dynamic programming. In *Mathematics and Computation in Music*, pages 199–204. Springer, 2015.

[24] Ben Wegbreit. The synthesis of loop predicates. *Communications of the ACM*, 17(2):102–113, 1974.

[25] Westley Weimer and George C Necula. Mining temporal specifications for error detection. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 461–476. Springer, 2005.

[26] Jinlin Yang, David Evans, Deepali Bhardwaj, Thirumalesh Bhat, and Manuvir Das. Perracotta: mining temporal api rules from imperfect traces. In *Proceedings of the 28th international conference on Software engineering*, pages 282–291. ACM, 2006.