MAIA Util: An NPM Package for Bridging Web Audio with Music-theoretic Concepts

Tom Collins[1,2]
[1]Music, Science and Technology Research Cluster
Department of Music
University of York, UK
tomthecollins@gmail.com

Christian Coulon[2]
[2]Music Artificial Intelligence Algorithms, Inc.
P.O. Box 73004
Davis, CA 95617
christianco@gmail.com

ABSTRACT

The Web Audio API and associated JavaScript packages have enabled developers to design interfaces where, to an unprecedented extent, the elements of music are at users' fingertips. When these interfaces are intended to help users to understand those musical elements, it can be useful to calculate or display manifestations of music-theoretic concepts, such as the key of an excerpt, the segmentation and labeling of chords, and melodic and harmonic intervals.

The MAIA Util package contains JavaScript code for executing these calculations. The input music representations are assumed to be symbolic, coming from MIDI or MusicXML files, or having been estimated from audio. This paper introduces the contents of the package and the music-cognitive research on which some of its constituent algorithms are based.

Some of the methods are of a more basic nature, such as for cyclic permutation of arrays or estimation of the pitch and octave of a note given its MIDI number and surrounding context. We have found use for these methods often enough during music interface development that they are included too.

The MAIA Util package is available for use from https://www.npmjs.com/package/maia-util

1. INTRODUCTION

In the domains of sound and interface design, packages such as Tone.js [25, 24] and NexusUI [3, 2] have increased the ease with which programmers can develop and prototype applications that make use of the Web Audio API [1, 29]. In the domain of music information retrieval (MIR), progress continues to be made with automatically analyzing and generating music-related data, especially with regards the training and testing of deep learning algorithms, and much of it implemented in open-source Python libraries. In the domain of computational music theory, a library called music21 [15] – also in Python – has proved popular for importing, displaying, and calculating features of symbolic music representations. All the while, the domain of music cognition proceeds with experimental investigation and computational modeling of how we perceive, create, and respond to music [17]. The broad premise of this paper is that while exciting progress is being made *within* each of these domains, exciting progress can also be made by improving the bridges that exist *between* them.

A more specific premise of the paper, and the MAIA Util package described below, is that often in sound and interface design, it can be useful to run and even display the results of MIR, music-theoretic, and music-cognitive algorithms. The remainder of the paper is structured around introducing and contextualizing implementations of some of these algorithms.

2. HELLO WORLD AND INPUT/OUTPUT

A MAIA Util "hello world" demo can be explored at https://tinyurl.com/y3rz5q73, and shows how the package can be used to analyze and display phenomena arising from an incoming symbolic representation, obtained from automatic transcription of the audio signal [18, 22]. For the sake of demonstrating this bridging of Web Audio, MIR, music theory, and music cognition, we show how MAIA Util can be used to calculate and graph the empirical distribution of estimated MIDI note numbers (MNN), as well as the distribution of estimated events throughout the average measure, quantized at the 16th-note level. Two of the MAIA Util functions used in this demo are `count_rows`, which is useful for calculation of possibly-multidimensional empirical distributions, and `bar_and_beat_number_of_ontime`, which takes an incrementing measure of time in a song (referred to as ontime) and an array consisting of the initial time signature and any subsequent changes, and returns the corresponding measure (bar) and beat numbers.

The input symbolic representation for the "hello world" demo is a custom-made, JSON format called a *Composition object*.[1] The exact format of the input is not of particular importance, however, since most of MAIA Util's methods operate on a point-set representation of the music,

$$E = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}, \tag{1}$$

with elements

$$\boldsymbol{e}_i = (x_i, y_i, z_i, w_i, u_i, v_i), \text{ where } i \in \{1, 2, \ldots, n\}, \tag{2}$$

and which is relatively straightforward to obtain from Node

[1]See https://crunchy.musicintelligence.co/composition/ for more details.

Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution**: owner/author(s).

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.

© 2019 Copyright held by the owner/author(s).

Package Manager (NPM) MIDI or XML parsers.[2] In terms of implementation, E is stored as a nested numeric JavaScript array. For a given point $\boldsymbol{e}_i$ as in (2),

- x_i is an *ontime*, which is an incrementing measure of time in a song or piece counted in quarter-note beats from 0 for measure 1 beat 1;
- y_i is an MNN, which is the numeric position of a key on the piano keyboard;
- z_i is a morphetic pitch number (MPN) [26], which is the numeric height of a note on the stave (see Section 5 for more details);
- w_i is a duration measured in quarter-note beats;
- u_i is a channel or staff number;
- v_i is a velocity in the range 0 (silent) to 1 (maximum loudness).

As mentioned in the introduction, the MAIA Util package is intended to bridge existing libraries built on the Web Audio API with both MIR algorithms (such as the audio-to-symbolic demo mentioned above) and music-theoretic concepts. As such, there is no one common or definitive output format. The guiding principle has been to make any array output convenient for subsequent use by JavaScript's built-in `map`, `filter`, and `reduce` methods, Tone.js' scheduling methods, NexusUI's GUI elements, and by the HTML5 canvas in general. The API can be explored at https://musicintelligence.co/api/maia-util/. It is documented with JSDoc and tested using Mocha. There is potential for optimization, and the approach at present is more functional than object-oriented.

The next three sections address the most important music-theoretic components of MAIA Util, while emphasizing that the applicability of these methods extends beyond purely music-theoretical concerns.

3. KEY ESTIMATION AND KEYSCAPES

A considerable amount of research in the music-cognitive literature has been devoted to the perception of key and tonality [14, 23]. Complementing this work are numerous efforts in the music-cognitive and music information retrieval literatures to automatically estimate the key of an excerpt from input audio or symbolic representations. While the use of key signatures in staff notation can be considered a music-theoretic concept, the above-mentioned work underlines the psychological reality or validity of key and tonality, as well as how the perceived key sometimes differs from the notated key. Therefore, being able to calculate, display, or otherwise-visualize the key of an excerpt in a web-based music interface can be useful, irrespective of whether that interface involves staff notation.

The key-estimation algorithms of Collins et al. [14] and Krumhansl [23] are based on listening studies involving the perception of tonality. The former operates directly on the audio signal, involves simulation of the inner ear and auditory cortex, and is implemented in Matlab, whereas the latter – known as the Krumhansl-Schmuckler key-finding algorithm – operates on symbolic input and involves correlating a pitch-class histogram of incoming music data with experimentally derived, idealized representations of the pitch-class content of each major and minor key. Due to its relative simplicity and speed, we implemented the latter algorithm in MAIA Util as the function `fifth_steps_mode`.

A common observation regarding musical structure – broadly construed – is that it is *hierarchical.* With regards tonality and key, this observation applies, say, to an excerpt that begins and ends in C major, with a modulation to G major partway through. On the highest level, it is accurate to state that the excerpt is in C major, but this overlooks the more fine-grained detail, where a modulation occurs from C to G and back again. Sapp [30] introduces the concept of a *keyscape*, which is a pyramidal representation of the output of a key-estimation algorithm that captures the hierarchical nature of tonality and key. Colored blocks toward the top of the pyramid represent key estimates of larger segments of an input excerpt, while blocks towards the base of the pyramid represent key estimates of smaller, more momentary segments of the excerpt. Key estimates of segments of equal length are represented by blocks in the same row, and, in the same row, a block to the left represents a segment that occurs earlier than one to the right. Keyscape calculation itself is not built into MAIA Util, but an interactive demo of keyscapes available from https://tinyurl.com/y5q8um4q demonstrates that it is relatively straightforward to calculate and display keyscapes based on the use of MAIA Util's `fifth_steps_mode`.

4. CHORD LABELING

Automatic chord labeling from input audio or symbolic representations is a problem of long and consistent interest in music computing [28, 6, 31, 16]. As a primary use case, the output of an accurate chord labeling system provides guitarists and other musicians a convenient means to begin playing along to and/or learning a song.[3] The function of chords in sequences is a topic that has received much attention from music theorists [8] and cognitive scientists [7, 21], but compared to work on key-estimation algorithms, there is less overlap between music-cognitive and MIR literatures when it comes to automatic chord labeling.

With measure-length granularity (or shorter), the lowest parts of a keyscape can be thought of as a chord-labeling system. For instance, if a measure (or less) of music were labeled as being in C major, then from a music-theoretic standpoint this would not be considered sufficient material to properly establish a key – rather it would probably contain some or all the pitch classes of the C-major triad and possibly some non-chord tones, i.e. a C-major chord. That is, the lowest parts of a keyscape provide chord-like labels.

What the keyscape does not provide explicitly, however, is a *segmentation* of the music. For instance, if an excerpt consisted of three-and-a-half measures articulating a C-major triad, followed by a change to a G-major triad for half a measure, then the correct segmentation and labeling would be (0, "C major"), (14, "G major"), where (x, l) denotes a chord label l beginning at ontime x. The information underlying the keyscape, however, would likely be (0, "C major"), (4, "C major"), (8, "C major"), (12, "C major"). That is, the first three-and-a-half measures are not combined into a single label, and the existence of the G-major chord is not

[2]E.g., https://www.npmjs.com/package/midiconvert and https://www.npmjs.com/package/xml2js

[3]E.g., https://chordify.net/

evident, with content in ontimes 12-14 (C major) and 14-16 (G major) being described by a single label ("C major").

Pardo and Birmingham [28] suggest that any viable chord analysis system ought to have both segmentation and labeling components. They propose a linear-time algorithm called HarmAn, which explores a subspace of all possible segmentations of an input symbolic representation, according to how well those segments score against predefined chord templates (pitch-class sets, such as $\{0, 4, 7, 10\}$ for "C 7"). Each predefined template has an associated label, and so the template giving rise to the maximum score for the segment under consideration will receive that label. Again giving priority to the algorithm's relative simplicity and speed, we implemented HarmAn in MAIA Util as the function `harman_forward`. The term "forward" refers to the subspace of all possible segmentations explored by the algorithm.

Partition point and *minimal segment* [28] are two important concepts involved in the early stages of HarmAn that have uses beyond chord labeling. We have implemented them in a corresponding function called `segment`. A partition point is an ontime in the input symbolic representation where a note either begins or ends. A minimal segment is the set of notes that sound between one partition point and the next. (One note may belong to more than one minimal segment.) With respect to HarmAn, the minimal segments are the building blocks of the segmentation. The scoring function favors the appending of two minimal segments if their combined score against the chord templates is greater than or equal to the sum of the scores for the two minimal segments considered in isolation. Returning to the example of three-and-a-half measures articulating a C-major triad, followed by a change to a G-major triad for half a measure, this process of considering combined or isolated minimal segments is how HarmAn would arrive at the correct segmentation and labeling of (0, "C major"), (14, "G major").

Beyond HarmAn, minimal segments can be used to:

- Provide a measure of monophony – the proportion of minimal segments in an excerpt that contain either one or zero note(s) can be used to measure the extent to which that excerpt is monophonic [9, 20];
- Extract the top (or bottom) line from polyphonic material, known as *skylining*;
- Group notes so as to retrieve all instances of a specified harmonic interval in an excerpt [9]. This is discussed further in the next section;
- Group notes so as to calculate empirical probability distributions for one or more excerpts, for analytic [12] or generative [13, 11] purposes.

5. MELODIC/HARMONIC INTERVALS

Identifying the interval between two consecutive notes (melodic interval) or between two notes that sound together (harmonic interval) is a fundamental topic in music theory. Quantitative and qualitative investigations of those intervals have been the source of much discourse. Representing a melody in relative (as intervals) rather than absolute (as note names or MIDI numbers) terms is also psychologically relevant and computationally advantageous: regarding psychological relevance, most listeners have relative rather than absolute means of pitch perception and production, i.e. we can recognize or attempt to sing "Happy birthday" irrespective of starting pitch; regarding computational advantage, most audio- and symbolic-based music recognition systems use differences between pitches (intervals) rather than the pitches themselves as a means of querying a database [5, 32].

For users with music-theoretic knowledge, a frustrating aspect of existing music-intervallic calculators is that the interval between the note pair (C4, F♯4) might be mislabeled as a diminished fifth, or vice versa that between the note pair (C4, G♭4) might be mislabeled as an augmented fourth, when the opposite is true. Both of these intervals are six MNNs, and if MNNs alone are used as the basis for the difference calculation, then both pairs are represented as (60, 66), so it is impossible to distinguish an F♯4 from a G♭4, and impossible to distinguish an augmented fourth from a diminished fifth.

There is a useful, complementary numeric representation of pitch called morphetic pitch number (MPN) [26] that helps to resolve this issue. Figure 1 contains some examples of (MNN, MPN)-pairs in the neighborhood of "middle C" $\equiv$ C4 $\equiv$ (60, 60). Whereas MNN is the numeric position of a key on the piano keyboard, MPN is the numeric height of a note on the stave. So while F♯4 and G♭4 map to the same key on the keyboard, they are notated on different lines of the stave, and so taking their MNNs and MPNs together, they are distinguishable: F♯4 maps to (MNN, MPN)-pair (66, 63), while G♭4 maps to (MNN, MPN)-pair (66, 64).[4]

Similarly, we can associate the difference between two (MNN, MPN)-pairs with correctly named musical intervals, meaning that entities such as "augmented fourth" and "diminished fifth" are distinguishable from one another. For example, two pitches p_1 and p_2, where p_1 is assumed lower in pitch than p_2, form the interval of a diminished fifth if and only if the difference between their corresponding (MNN, MPN)-pairs is (6, 4); they form the interval of an augmented fourth if and only if the difference between their corresponding (MNN, MPN)-pairs is (6, 3), and so on. The MAIA Util functions for converting between pitch and octave number, and (MNN, MPN)-space are called `pitch_and_octave2midi_note_morphetic_pair` and `midi_note_morphetic_pair2pitch_and_octave`.

Identifying all instances of a specified harmonic interval is slightly more difficult than the corresponding task for a melody, because (a) two notes forming the relevant interval may not begin at the same time, even though this is how we typically think of rudimentary harmonic intervals, and (b) there may be notes that occur in between them in a chord, meaning that we cannot rely on comparisons between adjacent notes when the notes are placed in ascending order. Problem (a) can be solved with use of `segment` (see previous section), because if two notes do form the specified harmonic interval, they will occur together in at least one minimal segment. Problem (b) can be solved as follows:

1. Let us notate the point-set representation of notes belonging to this minimal segment as $E = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$, and then let us project this set down to a space where only the dimensions of MNN and MPN are retained, notating this projected set as $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_m\}$, where m may be less than n. So

[4]In mathematical terms, we say there is a bijection between pitch and octave number, and (MNN, MPN)-space.

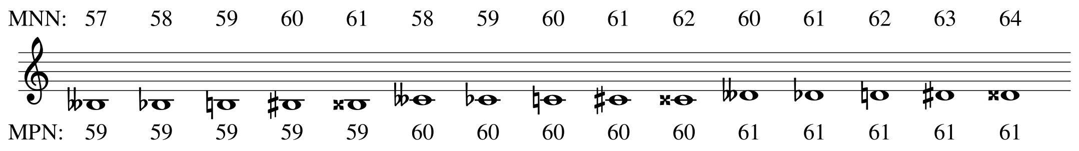
MNN: 57 58 59 60 61 58 59 60 61 62 60 61 62 63 64
MPN: 59 59 59 59 59 60 60 60 60 60 61 61 61 61 61

Figure 1: Some MIDI note numbers (MNN) and morphetic pitch numbers (MPN) close to C4 ("middle C").

$\boldsymbol{d}_i = (y_i, z_i)$ for some MNN y_i and MPN z_i;

2. Let $\boldsymbol{v} = (v_y, v_z)$ be the (MNN, MPN)-difference corresponding to the specified harmonic interval. Then the specified harmonic interval occurs in this minimal segment if and only if there exist some $i, j \in \{1, 2, \ldots, m\}$ satisfying $\boldsymbol{d}_i + \boldsymbol{v} = \boldsymbol{d}_j$. In other words, $\boldsymbol{d}_i$ translates to $\boldsymbol{d}_j$ by the vector $\boldsymbol{v}$.

3. Meredith [27] defines the *maximal translatable pattern* (MTP) of the vector $\boldsymbol{v}$ in the point-set D as the set of all points in D that are translatable to another point in D by the vector $\boldsymbol{v}$, and MAIA Util contains an implementation of this concept as the function `maximal_translatable_pattern`. So we can solve problem (b) by calculating E, D as described above and then seeing if the array returned by `maximal_translatable_pattern(v, D)` is populated (specified interval occurs) or empty (specified interval does not occur).

Algorithms for calculating MTPs in an unsupervised manner form the basis of geometric pattern discovery in music, enabling the discovery of repeated elements such as riffs, motifs, themes, and sections [27, 10]. A discussion of these algorithms is beyond the scope of this paper, but their representational and functional bases are present in MAIA Util.

6. UTILITY FUNCTIONS

This section addresses a couple of methods that are very widely used in our interfaces, but not as complex or closely connected to music cognition as those described above.

6.1 Pitch estimation

During the development of several web-based music interfaces, we have found it necessary to estimate the pitch of given MNNs in a surrounding musical context. Such a need tends to arise because MIDI is a ubiquitous input format, but lacks accurate pitch and octave information – e.g., is MNN 66 an F♯4 or G♭4? – and we strive for music-theoretic correctness. Just as it is frustrating when an interval is mislabeled (see the discussion of diminished fifths and augmented fourths in the previous section), so it annoys musically trained users when pitch sequences such as (C, D, E, G♭, G) appear in an interface, which probably ought to be (C, D, E, F♯, G). These misspelt sequences tend to result from oversimple MNN-to-pitch estimation algorithms. From an educational perspective, mislabeling of pitches and intervals is also detrimental to novice users who are learning, implicitly or explicitly, about music theory via use of an interface.

A recent discussion about pitch estimation from MNN on the music21 Google Group [4] underlines that music21 does not have built-in functionality for addressing this problem. The solution suggested on the forum involves a base-40 numeric representation [19], which has less psychological validity compared to the solution involving MNN-MPN space, where MNNs model our perception of chromatic pitch and MPNs model our perception of diatonic pitch.

MAIA Util contains a function called `guess_morphetic`, which works by providing an MNN and the key of an excerpt (see discussion of `fifth_steps_mode` above), and then mapping the provided MNN to the (MNN, MPN)-pair that is most likely to occur in the key. For instance, F♯'s are generally more numerous in pieces in C major than are G♭'s, so if the provided MNN and key are 66 and C major, respectively, then 66 will map to the (MNN, MPN)-pair (66, 63) for F♯ and MPN 63 will be returned, rather than mapping to the (MNN, MPN)-pair (66, 64) for G♭ and returning MPN 64. Due to the bijection between pitch and octave, and (MNN, MPN)-space (see Section 3), once we have the (MNN, MPN)-pair, we can map unambiguously to a pitch and octave using `midi_note_morphetic_pair2pitch_and_octave`. That is, (MNN, MPN)-pair (66, 63) maps to F♯4. This is how MAIA Util can be used to convert an input MNN sequence (60, 62, 64, 66, 67) to the correctly spelled pitch and octave sequence (C4, D4, E4, F4♯, G4).

Pitch estimation algorithms in the literature tend to be more complex than that based on MAIA Util's `guess_morphetic` (e.g., [26]). Informally, we have not observed a big tradeoff in accuracy, however, and so favor a simple and speedy approach at present.

6.2 Projection, sorting, and deduplication of nested numeric arrays

Since point-set representations featured prominently in Section 2 and have been present in other sections too, it seems appropriate to conclude the tour of MAIA Util with some of the point set-specific functionality. There is a function `orthogonal_projection_not_unique_equalp` that can retain/remove a specifiable selection of dimensions from an input point set, which is a form of *projection* operation in mathematics. The sorting of a point set can be achieved with `sort_rows`, which comprises calling the built-in `sort` method with a helper function `lex_more` that returns +1 if the first input is *lexicographically more* than the second input, and −1 otherwise. The `sort_rows` function returns both the sorted array as well as the indices of the elements from the input array. Another function, `unique_rows`, returns a sorted, deduplicated version of the input array, as well as the indices of the elements from the input.

7. DISCUSSION

Considerable progress has been made in recent years concerning complex topics and problems within the domains of web audio, MIR, music theory, and music cognition. In some cases, this progress has manifested in usable, if circumscribed, algorithmic solutions. The current paper represents a modest attempt to bring together some of these algorithms and associated insights. It describes an NPM package called MAIA Util that contains implementations of methods that allow the algorithms to interface with each other. One outcome consists of interactive demos that bridge the above-mentioned domains in potentially novel and interesting ways.

Through introducing some of MAIA Util's methods and use cases for key estimation, chord labeling, and interval identification, we have suggested that it can be useful to calculate and sometimes display manifestations of music-theoretic concepts, even if the interface that displays these manifestations is not intended for exploring music theory or staff notation per se. We have also emphasized how it can be frustrating and potentially detrimental to users when music interfaces show "unmusical" pitch spellings or intervals, and we have demonstrated how our use of MNN-MPN space helps to address some of these issues. Where our key estimation, chord labeling, and interval identification algorithms draw on findings from the literature on music cognition, this has been pointed out because it is reasonable to assume that use of algorithms for which there is some underlying psychology validity will lead to calculations and/or visualizations that listeners find to be more convincing or realistic.

Up until now, our criterion for inclusion of a method in MAIA Util has been that the method is required in multiple, in-development music interfaces. Future work is likely to consist of including more algorithms for quantization, score following, pattern discovery, and music generation. Whether these algorithms warrant their own packages – either because they are too complex or because they are not widely required – or are added to MAIA Util remains to be seen. Either way, there is something exciting about the potential for wide and rapid dissemination afforded by the web, combined with contributions from several different intersecting music research domains that are coalescing in web audio space.

8. ACKNOWLEDGMENTS

We are grateful to Jeremy Yu, with whom the first author worked on a framework for running multiple MIR algorithms and integrating their results into Composition objects. This work formed the basis of the "Hello world" demo.

9. REFERENCES

[1] P. Adenot, R. Toy, C. Wilson, and C. Rogers. Web Audio API. https://www.w3.org/TR/webaudio/. Retrieved November 10, 2017.

[2] J. Allison, W. Conlin, D. Holmes, Y. Oh, and B. Taylor. NexusUI GitHub repository. https://github.com/nexus-js/ui/. Retrieved August 5, 2015.

[3] J. Allison, W. Conlin, D. Holmes, Y. Oh, and B. Taylor. Simplified expressive mobile development with NexusUI, NexusUp and NexusDrop. In *Proceedings of the New Interfaces for Musical Expression Conference*, 2014.

[4] T. Anders and C. S. Sapp. Converting MIDI pitch or pitch class numbers into enharmonically most likely pitches depending on a key. https://tinyurl.com/y232n8jw. Retrieved June 5, 2019.

[5] A. Arzt, S. Böck, and G. Widmer. Fast identification of piece and score position via symbolic fingerprinting. In *Proceedings of the International Society for Music Information Retrieval Conference*, pages 433–438, 2012.

[6] J. P. Bello and J. Pickens. A robust mid-level representation for harmonic content in music signals. In *Proceedings of the International Society for Music Information Retrieval Conference*, pages 304–311, 2005.

[7] E. Bigand, B. Poulin, B. Tillmann, F. Madurell, and D. A. D'Adamo. Sensory versus cognitive components in harmonic priming. *Journal of Experimental Psychology: Human Perception and Performance*, 29(1):159–171, 2003.

[8] W. E. Caplin. *Analyzing classical form: an approach for the classroom*. Oxford University Press, New York, NY, 2013.

[9] T. Collins. Stravinsqi/De Montfort University at the MediaEval 2014 C@merata task. In *Proceedings of the MediaEval Workshop*, 2014.

[10] T. Collins, A. Arzt, H. Frostel, and G. Widmer. Using geometric symbolic fingerprinting to discover distinctive patterns in polyphonic music corpora. In *Computational Music Analysis*, pages 445–474. Springer, 2016.

[11] T. Collins and R. Laney. Computer-generated stylistic compositions with long-term repetitive and phrasal structure. *Journal of Creative Music Systems*, 1(2), 2017.

[12] T. Collins, R. Laney, A. Willis, and P. H. Garthwaite. Modeling pattern importance in Chopin's mazurkas. *Music Perception*, 28(4):387–414, 2011.

[13] T. Collins, R. Laney, A. Willis, and P. H. Garthwaite. Developing and evaluating computational models of musical style. *AI EDAM*, 30(1):16–43, 2016.

[14] T. Collins, B. Tillmann, F. S. Barrett, C. Delbé, and P. Janata. A combined model of sensory and cognitive representations underlying tonal expectations in music: From audio signals to behavior. *Psychological Review*, 121(1):33–65, 2014.

[15] M. S. Cuthbert and C. Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. In *Proceedings of the International Society for Music Information Retrieval Conference*, pages 637–642, 2010.

[16] W. B. De Haas, J. P. Magalhães, F. Wiering, and R. C. Veltkamp. Automatic functional harmonic analysis. *Computer Music Journal*, 37(4):37–53, 2013.

[17] D. Deutsch. *The psychology of music*. Academic Press, San Diego, CA, 3rd edition, 2013.

[18] C. Hawthorne, E. Elsen, J. Song, A. Roberts, I. Simon, C. Raffel, J. Engel, S. Oore, and D. Eck. Onsets and frames: Dual-objective piano transcription. *arXiv preprint arXiv:1710.11153*, 2017.

[19] W. B. Hewlett. A base-40 number-line representation of musical pitch notation. *Musikometrika*, 4:1–14, 1992.

[20] B. Janssen, T. Collins, and I. Ren. Algorithmic ability to predict the musical future: Datasets and evaluation. In *Proceedings of the International Society for Music Information Retrieval Conference*. In press.

[21] S. Koelsch. Music-syntactic processing and auditory memory: Similarities and differences between eran and mmn. *Psychophysiology*, 46(1):179–190, 2009.

[22] F. Krebs, S. Böck, and G. Widmer. Rhythmic pattern modeling for beat and downbeat tracking in musical audio. In *Proceedings of the International Society for Music Information Retrieval Conference*, pages 227–232, 2013.

[23] C. L. Krumhansl. *Cognitive foundations of musical pitch*. Oxford University Press, New York, NY, 1990.

[24] Y. Mann. Tone.js GitHub repository. https://github.com/Tonejs/Tone.js. Retrieved August 5, 2015.

[25] Y. Mann. Interactive music with Tone.js. In *Proceedings of the Web Audio Conference*, January 2015.

[26] D. Meredith. The ps13 pitch spelling algorithm. *Journal of New Music Research*, 35(2):121–159, 2006.

[27] 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.

[28] B. Pardo and W. P. Birmingham. Algorithms for chordal analysis. *Computer Music Journal*, 26(2):27–49, 2002.

[29] C. Rogers. Web Audio API. https://www.w3.org/TR/2012/WD-webaudio-20121213/. Retrieved November 10, 2017.

[30] C. S. Sapp. Visual hierarchical key analysis. *ACM Computers in Entertainment*, 3(4):1–19, 2005.

[31] D. Tymoczko. Review of Michael Cuthbert, music21: A toolkit for computer-aided musicology (http://web.mit.edu/music21/). *Music Theory Online*, 19(3), 2013.

[32] A.-C. Wang and J. Smith. System and methods for recognizing sound and music signals in high noise and distortion, 2012. Patent US 8,190,435 B2. Continuation of provisional application from 2000.