

MAPPING MIDI TO THE SPIRAL ARRAY: DISAMBIGUATING PITCH SPELLINGS

E. Chew*

*University of Southern California, Integrated Media Systems Center
Daniel J. Epstein Department of Industrial and Systems Engineering
3715 McClintock Avenue GER240 MC:0193, Los Angeles CA 90089-0193, USA.
echew@usc.edu*

Y.-C. Chen

*University of Southern California, Integrated Media Systems Center
Daniel J. Epstein Department of Industrial and Systems Engineering
yunchinc@usc.edu*

Abstract The problem of assigning appropriate pitch spellings is one of the most fundamental problems in the analysis of digital music information. We present an algorithm for finding the optimal spelling based on the Spiral Array model, a geometric model embodying the relations in tonality. The algorithm does not require the key context to be determined. Instead, it uses a center of effect (c.e.), an interior point in the Spiral Array model, as a proxy for the key context. Plausible pitch spellings are measured against this c.e., and the optimal pitch is selected using the nearest neighbor criteria. Two examples are given from Beethoven's Sonata Op. 109 to illustrate the algorithm. The algorithm is implemented and the results used in MuSA – a music visualization software using the Spiral Array. We present and analyze computational results from test runs on MIDI files of two movements from Beethoven's Piano Sonatas Op.79 and Op.109.

Keywords: Music Information Processing, Pitch Spelling, Content Extraction and Analysis, Visualization, Knowledge Representation, Pattern Recognition, Artificial Intelligence, Algorithm Design.

*Partial funding provided by a Women In Science and Engineering grant and by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center Cooperative Agreement No. EEC-9529152.

Introduction

The problem of assigning appropriate pitch spellings to digital music information is endemic to any music analysis or recognition system. We propose an algorithm that can assign pitch spellings that are consistent with the key context of the passage using the Spiral Array model (Chew, 2000). We implement the algorithm in **MuSA** – a software package for MUsic visualization using the Spiral Array. We illustrate the algorithm using two excerpts from Beethoven’s Piano Sonata Op.109. The algorithm is then tested on MIDI files of the first movement of Beethoven’s Piano Sonata Op.109 and the third movement of his Piano Sonata Op.79.

Pitch spelling is a problem that is endemic to any music analysis or recognition system, and is an artifact of equal temperament tuning in western tonal music. In an equal tempered system, several pitches are approximated by the same frequency so as to ensure that music in different keys can be played using a finite pitch set. Each pitch in the set corresponds to several pitch names; the name reveals its key context and determines its notation.

MIDI and other digital audio file formats typically assign a numerical value to each pitch without regard to the key context. The key of a piece of music determines the note material of that piece, and hence the spellings for these pitches. The correct spelling of the pitches, in turn, serve as clues to the identity of the key.

Our knowledge-based method uses a nearest-neighbor approach on the Spiral Array model to find the optimal pitch spelling. We do not use information about the key as a pre-cursor to assigning pitch names. Cumulative information embodied in a spatial point, the center of effect, acts as a proxy for the key context. Special attention is paid to the initial chunk of music prior to the establishing of a tonal context, and pitch assignments are re-visited so as to align them with the context.

The Spiral Array is a spatial model for tonality whose geometry reflects perceived relations among the represented entities. In the model, musical entities are represented in the interior of a spiral array; higher level entities are successively defined as convex combinations of their lower level parts. This interior point approach to modeling music information has been found to be particularly effective in the computational modeling of key-finding (Chew, 2001) and determining modulations (Chew, 2002).

An important feature of the Spiral Array model is that points in space labeled by pitch names from the same key form a compact cluster. We exploit this fact to present a nearest neighbor approach to finding the best spelling for each pitch in the piece without having to first ascertain

the key context. The Spiral Array offers an effective alternative to other pitch spelling approaches such as, the preference rule approach posed by Temperley (2002) and the interval optimization approach proposed by Cambouropoulos (2001).

An overview of the Spiral Array model is given in Section 1. A description of the problem of pitch spelling and the proposed algorithm follows in Section 2. The next section describes the music visualization software that uses this algorithm to assign and revise the pitch spellings so as to accurately map MIDI data to their appropriate representations on the Spiral Array and to calculate the most likely key.

The ability to correctly assign pitch spellings that are consistent with the key context will add accuracy and reliability to computer systems for music analysis, automated transcription and content-based music information retrieval. Rowe (2001) shows that two of the fundamental components in machine musicianship processes are pitch and time structures. Figure 1 shows the role of pitch spelling in the context of music content extraction and analysis.

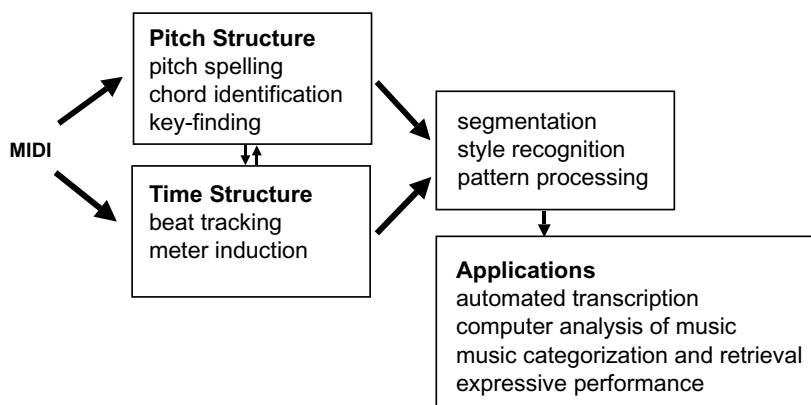


Figure 1. Music Information Content Extraction and Analysis

1. The Spiral Array Model

This section describes the representation on which the pitch spelling algorithm is based. The Spiral Array model is a mathematical model for tonality, representing musical entities such as pitches, chords and keys and the relations among them. It is a geometric model that is configured so that closely related musical entities are positioned in compact clusters. Chords are generated from their component pitches as convex combinations of the pitch positions; and, keys are generated from

their defining chords in a similar manner. Please refer to the end of this chapter for a glossary of musical terms.

1.1 Spatial Representation of Tonal Entities

In the Spiral Array, pitches are represented at each quarter turn of an ascending spiral, and neighboring pitches are five scale steps (that is to say, the distance of a Perfect fifth) apart, which results in vertically aligned pitches being three major scale steps apart:

$$\mathbf{P}(k) \stackrel{\text{def}}{=} \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} r \sin \frac{k\pi}{2} \\ r \cos \frac{k\pi}{2} \\ kh \end{bmatrix}.$$

Each pitch is indexed by its distance, according to the number of Perfect fifths, from an arbitrarily chosen reference pitch, C.

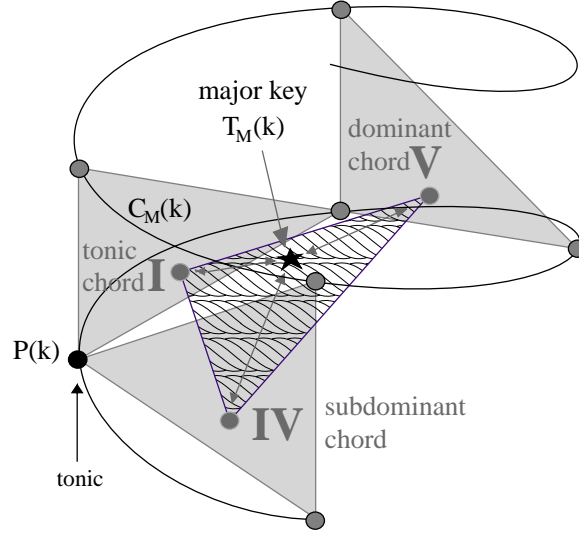


Figure 2. Geometric representation of a major key in the Spiral Array.

Major and minor chords are represented as convex combinations of their component pitch positions, and the weights on the three pitches are constrained to be monotonically non-decreasing according to the importance of that pitch:

$$\mathbf{C}_M(k) \stackrel{\text{def}}{=} w_1 \cdot \mathbf{P}(k) + w_2 \cdot \mathbf{P}(k+1) + w_3 \cdot \mathbf{P}(k+4),$$

where $w_1 \geq w_2 \geq w_3 > 0$ and $\sum_{i=1}^3 w_i = 1$;

$$\mathbf{C}_m(k) \stackrel{\text{def}}{=} u_1 \cdot \mathbf{P}(k) + u_2 \cdot \mathbf{P}(k+1) + u_3 \cdot \mathbf{P}(k-3),$$

where $u_1 \geq u_2 \geq u_3 > 0$ and $\sum_{i=1}^3 u_i = 1$.

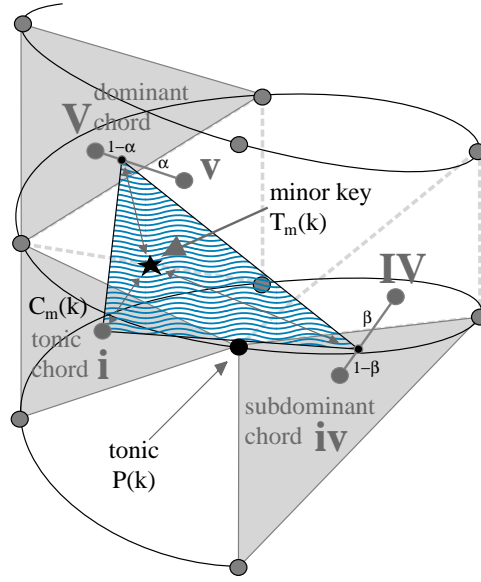


Figure 3. Geometric representation of a minor key in the Spiral Array.

Major and minor key representations are specified as convex combinations of their defining chords. Again, the weights are constrained to be monotonically non-decreasing according to the importance of the chord. The minor key definition is slightly more involved because it uses both major and minor versions of the chords indexed by $(k-1)$ and $(k+1)$. The additional parameters α and β indicate the importance of the major vs. minor versions of the chords. Figures 2 and 3 show the derivation of the major and minor chord representations respectively.

$$\mathbf{T}_M(k) \stackrel{\text{def}}{=} \omega_1 \cdot \mathbf{C}_M(k) + \omega_2 \cdot \mathbf{C}_M(k+1) + \omega_3 \cdot \mathbf{C}_M(k-1),$$

where $\omega_1 \geq \omega_2 \geq \omega_3 > 0$ and $\sum_{i=1}^3 \omega_i = 1$.

$$\mathbf{T}_m(k) \stackrel{\text{def}}{=} v_1 \cdot \mathbf{C}_m(k) + v_2 \cdot [\alpha \cdot \mathbf{C}_M(k+1) + (1-\alpha) \cdot \mathbf{C}_m(k+1)]$$

$$\begin{aligned}
& +v_3 \cdot [\beta \cdot \mathbf{C}_m(k-1) + (1-\beta) \cdot \mathbf{C}_M(k-1)], \\
\text{where} \quad & v_1 \geq v_2 \geq v_3 > 0 \text{ and } v_1 + v_2 + v_3 = 1, \\
\text{and} \quad & 0 \geq \alpha \geq 1, 0 \geq \beta \geq 1.
\end{aligned}$$

1.2 Calibrating Model Distances

A design criteria of the Spiral Array model is that spatial distances should mirror perceived relations among the represented entities. The selection of the weights and parameters are based on distance constraints generated by music knowledge. The knowledge incorporated into the model can be summarized in the following list:

- 1 Certain distance relations between pairs of pitches sound more stable (and hence closer) than others. Closer interval relations should be represented correspondingly by shorter inter-object distances.
- 2 Some pitches in a chord are more important than others in relation to the chord, and all other pitches are less closely related to the chord. The selection of the chord weights should result in distance relations that reflect the perceived relations.
- 3 Some pairs of pitches, when sounded in succession, strongly indicate the identity of a key. These pitch pairs should be closer to the key they indicate than others. We use two such pitch pairs to calibrate the distance relations.

Feasible weights and parameters are found by both analytical and numerical means. Details on the calibration of the model can be found in Chew (2000).

1.3 Generating a Center of Effect

In the Spiral Array model, any collection of notes (also known as pitch events) generates a center of effect (c.e.). For a sequence of music time series data, the c.e. is a point in the interior of the Spiral Array that is the convex combination of the pitch positions weighted by their respective durations. If \mathbf{p}_i represents the position of the i -th pitch class in the Spiral Array, and d_i represents its duration, the sequence of notes in a melody up to the t -th pitch event can be written as $\{(\mathbf{p}_i, d_i) : i = 1 \dots t\}$. The center of effect of this pitch collection is defined as:

$$\mathbf{c}_t \stackrel{\text{def}}{=} \sum_{i=1}^t \frac{d_i}{D_t} \cdot \mathbf{p}_i, \quad \text{where } D_t = \sum_{i=1}^t d_i.$$

This method of generating c.e.'s was incorporated into the Center of Effect Generator (CEG) algorithm for key-finding described in Chew (2001). At each time step, a c.e. is calculated based on the cumulative pitch and duration information. The sequence of c.e.'s create a trajectory that gravitates quickly toward the spatial representation of the key context. The closest key is found through a nearest neighbor search:

$$\arg \min_{T \in \mathbf{T}} \|\mathbf{c} - T\|,$$

where $\mathbf{T} = \{\mathbf{T}_m(k) \forall k\} \cup \{\mathbf{T}_M(k) \forall k\}$ is a finite collection of keys represented in the Spiral Array, both major and minor. The Spiral Array model was shown to be more efficient and accurate in identifying the most likely key than existing models for key-finding.

2. Pitch Spelling

The problem of pitch spelling arises because the same pitch, that is to say, sound of a given frequency, can be assigned different names based on the key context. Pitches of the same frequency but denoted by different pitch names are said to be enharmonically equivalent. In a MIDI file or other digital music formats, a pitch is represented by a numerical value indicating its frequency and not its letter name. Each pitch can correspond to more than one letter name, and this letter name is dependent on the key context. The name of the pitch also determines the notation and serves as a clue to the key context. To solve the pitch spelling problem is to determine the appropriate name for the pitch that is consistent with the key context.

Pitches in a given key occupy a compact space in the Spiral Array model. Thus, the problem of finding the best pitch spelling corresponds to finding the pitch representation that is nearest to the current key context. It has been shown in Chew (2001) that the cumulative c.e., over the course of the piece, gravitates quickly toward the spatial point representing its key context. We use the evolving c.e. as a proxy for the key context, without having to determine the actual key. Each plausible spelling of the new pitch is measured against the current c.e., and the pitch that satisfies the nearest neighbor criteria is selected to be the appropriate pitch name.

More concretely, each pitch read from the MIDI file will correspond to two or three most probable letter names as shown (the numbers in the brackets indicate the index of the pitch in the Spiral Array):

<i>Row</i>	<i>Spelling 1</i>	<i>(index)</i>	<i>2</i>	<i>(index)</i>	<i>3</i>	<i>(index)</i>
0	B♯	(12)	C	(0)	Dbb	(-12)
1	C♯	(7)	D♭	(-5)	B♯♯	(19)
2	C♯♯	(14)	D	(2)	Ebb	(-10)
3	D♯	(9)	E♭	(-3)	Fbb	(-15)
4	D♯♯	(16)	E	(4)	F♭	(-8)
5	E♯	(11)	F	(-1)	Gbb	(-13)
6	E♯♯	(18)	F♯	(6)	G♭	(-6)
7	F♯♯	(13)	G	(1)	Abb	(-11)
8	G♯	(8)	A♭	(-4)		
9	G♯♯	(15)	A	(3)	Bbb	(-9)
10	A♯	(10)	B♭	(-2)	Cbb	(-14)
11	A♯♯	(17)	B	(5)	C♭	(-7)

Notice that the index triplets are of the form

$$\langle index - 12, index, index + 12 \rangle .$$

The Spiral Array can easily be extended to include indices of larger magnitude, such as pitch index -16 or 20 for Row 8.

Suppose that at time t , n_t notes have been sounded as indicated in the MIDI file. Let $index(i)$ be the pitch index closest to 0 in the row corresponding to the i -th pitch, $i = 1 \dots n_t$. This choice will bias the notation towards fewer sharps (♯) and flats (♭). The most probable pitch name assignments are given by the triplet:

$$\langle index(i) - 12, index(i), index(i) + 12 \rangle .$$

One could certainly extend the probable names to more than three, but three is sufficient in practice.

We re-define \mathbf{c}_t to represent the center of effect of all pitch events up to and including time t :

$$\mathbf{c}_t \stackrel{\text{def}}{=} \sum_{i=1}^{n_t} \frac{d_i}{D_t} \cdot \mathbf{p}_i, \quad \text{where } D_t = \sum_{i=1}^{n_t} d_i.$$

This definition generalizes the earlier one in Section 1 to allow more than one pitch event to take place at any given time.

In the Spiral Array, the pitch index reveals the pitch spelling. Hence, for any given pitch, the index that is consistent with the key context is given by:

$$index^* = \arg \min \left\{ \begin{array}{l} \|\mathbf{P}(index(i) - 12) - \mathbf{c}_t\|, \\ \|\mathbf{P}(index(i)) - \mathbf{c}_t\|, \\ \|\mathbf{P}(index(i) + 12) - \mathbf{c}_t\| \end{array} \right\}$$

We segment the MIDI data into chunks for analysis, so as to batch process the pitch name assignments.

The implementation of this method of assigning pitch names is only problematic at the beginning of the piece. At the beginning, there exists no hypothesis for the key and no center of effect has yet been generated. We propose the following scheme for initializing the algorithm:

- 1 The notes in the first chunk are assigned indices closest to 0¹;
- 2 a c.e. is generated based on this assignment; and,
- 3 the original assignments are re-visited to make them consistent with the key context.

3. MuSA: A Music Visualization Software

MuSA is a music visualization software based on the Spiral Array model. The software reads MIDI files, extracts the content and displays note and key information in real time. MuSA is developed using Java2 SDK, Standard Edition, with Java3D API. The Java classes parallel the definitions of the Spiral Array model described in Section 1.

The three-dimensional structure of the Spiral Array and the clustering of musical entities linked by meaningful tonal relations lends itself easily to the visualization of tonal patterns in music information. The model creates objects in space that represent musical entities, and is particularly suited to an object-oriented implementation.

In MuSA, music sequences defined by the MIDI file is played and analyzed in real time. Pitches map to positions in the Spiral Array model, shown on the left-hand-side of the display panel portrayed in Figure 4. The appropriate pitch name is necessary to ensure a mapping to the correct position. The algorithm described in Section 2 is incorporated into this part of MuSA.

Each pitch is represented by a colored ball with radius corresponding to its cumulative durations. There are a total of 35 possible colored spheres on the spiral in the display panel, representing the pitches F \flat through B \sharp . The user can adjust the radius, vertical step and view angle of the Spiral Array in real time.

The distance of the c.e to each major and minor key is graphed on the bar charts on the right-hand-side of the display panel. Like the pitches on the Spiral Array, adjacent keys on the bar chart are five scale steps apart. This arrangement produces a smoother distance profile for

¹The choice of indices close to 0 will bias the notation towards fewer sharps and flats.

easier viewing. The distance to major keys (labeled by capital letter key names) is graphed on the upper bar chart, and the distance to minor keys (labeled by small letter key names) on the lower bar chart.

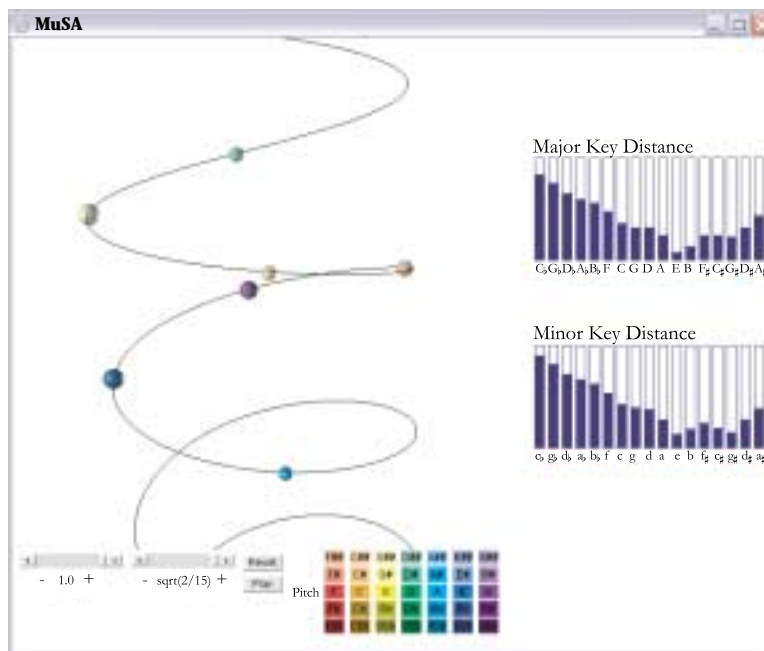


Figure 4. MuSA – MUsic visualization using the Spiral Array – display panel after eight bars of the Beethoven Op.109 first movement example.

4. Illustrative Examples: Beethoven Op.109

We first illustrate the algorithm using the opening bars of Beethoven's Piano Sonata Op.109 (see Figure 5). The musical segment was saved as a MIDI file and given as input to the MuSA software.



Figure 5. Opening bars of Beethoven's Piano Sonata, Op.109 (Example 1)

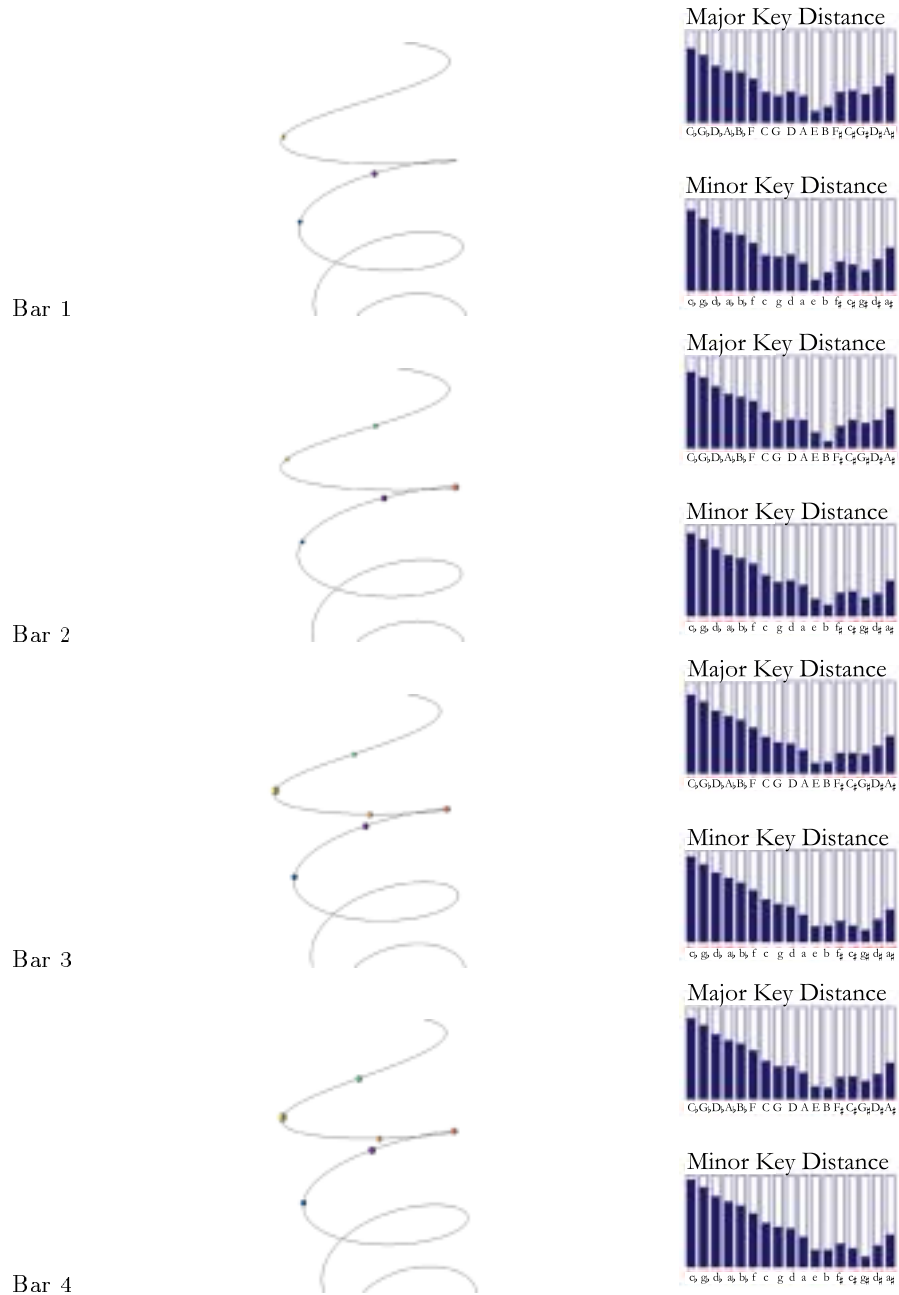


Figure 6. Pitch clusters and key charts for the Beethoven Op.109 sequence (b.1-4)

The MIDI file was segmented into one-bar chunks each. As the MIDI file is played, the display is updated at each chunk. The salient parts of the display corresponding to the first four chunks are shown in Figure 6. From the very first chunk, E major emerges as the most likely key with B major as a close second.

Prior to implementing the initializing steps outlined at the end of Section 2, the algorithm spelled the first note as $A\flat$ rather than the correct $G\sharp$, then proceeded to assign the correct spellings. The first error occurred because the algorithm required an index closest to 0. The index for $A\flat$ is closer to 0 than that for $G\sharp$. By re-visiting the first chunk and aligning the pitch spellings to the c.e. generated, the error was eliminated.



Figure 7. Bars $25\frac{1}{2} - 33\frac{1}{2}$ of Beethoven's Piano Sonata, Op.109 (Example 2)

The initial bars of the movement are relatively straightforward. Next, we jump ahead to the tonally more unstable bars $25\frac{1}{2} - 33\frac{1}{2}$ shown in Figure 7. Java's default MIDI-to-text converter produces spellings from the pitch set for $D\flat$ minor and $A\flat$ minor. We compare this solution to the spelling prescribed by the pitch spelling algorithm with C major as the initial c.e., and with $G\sharp$ minor (the key context of the preceding section) as the initial c.e.

default	{ B, G, $B\flat$, $E\flat$, $A\flat$, $D\flat$, G \flat , C \flat }
spelling (C)	{ B, G, $B\flat$, $E\flat$, $A\flat$, $D\flat$, G \flat , C \flat }
spelling ($g\sharp$)	{ B, $F\sharp$, $C\sharp$, $G\sharp$, $D\sharp$, $A\sharp$, $F\sharp\sharp$, $A\sharp\sharp$ }

This second example shows clearly that the results of the algorithm with $G\sharp$ minor as the initial c.e. produces the spelling closest to the annotated score. The only error, $A\sharp\sharp$, occurs once. With the incorrect initial c.e., the pitch spelling algorithm behaves no better than the

default spelling prescribed by Java’s MIDI-to-text converter. Evidence suggests that the correct c.e. is the reason the pitch spelling algorithm behaves better than the default.

5. Computational Results

In this section, we describe the results when the algorithm was tested on the entire third movement of Beethoven’s Piano Sonata No. 25 in G Major, Op.79 (1809), and the first movement of his Piano Sonata No. 30 in E Major, Op.109 (1820). The scores were scanned into the Sibelius music notation program and checked visually and aurally for optical recognition errors. MIDI files were generated from the scores for analysis. In MuSA, the MIDI files were segmented into beat-sized chunks and each note assigned a spelling using the algorithm proposed in Section 2.

The two Beethoven examples were chosen because even though their themes follow identical harmonically patterns, the two pieces evolve in entirely different ways. The Op.79 sonata was composed in the middle period and the Op.109 in the late period of Beethoven’s musical career. This translates to more predictable tonal patterns in the Op.79 movement than in the one from Op.109. The increased tonal complexity of the Op.109 example over the Op.79 example is reflected in the computational results compiled in the table below. The Op.79 example only had one error, while the Op.109 example had 73. The percentage of correct spellings were 99.93% and 95.18% respectively, with an overall rate of 97.44%.

<i>Piece</i>	<i>No. of notes²</i>	<i>No. correctly spelt</i>	<i>% correct</i>
Beethoven Op.79 (3rd mvt)	1375	1374	99.93
Beethoven Op.109 (1st mvt)	1516	1443	95.18
TOTAL	2891	2817	97.44

Cambouropoulos (2001) tested his algorithm on eight Mozart Piano Sonatas (K279-K283 and K331-K333). He calculated the percentage of correctly spelled notes based only on notes with accidentals in order to account for his algorithm’s preference for minimal use of accidentals. Variants of the algorithm were tested on the eight Mozart sonatas, achieving correct spelling rates between 94% and 96.2%.

The algorithm proposed in this paper incorporates a preference for a lack of accidentals only in the initial conditions. Following the first step, the algorithm gives preference to spellings close to the key con-

text as represented by the c.e. Hence, the percentage correct rates are calculated for all notes, with or without accidentals. Assuming that, in Cambouropoulos (2001), all the notes without accidentals were spelt correctly, his overall correct spelling rates ranged between 98.38% and 98.94%.

In order to compare the two algorithms, further tests need to be performed using the same test set. Cambouropoulos' test set, the selected Mozart sonatas, were composed between 1775 and 1784. The Beethoven sonatas we used were composed in 1809 and 1820 respectively. Beethoven took Mozart's harmonic language a step further and expanded on the tonal palette of the classical period. His late piano sonatas, especially, are noted for their ingenious tonal patterns, which are more difficult to track computationally.

5.1 Error Analysis

In the computational tests, there were basically two types of errors. One resulting from the algorithm's ignorance of linear motion, and the other from insufficient sensitivity to key changes. The tonal system prioritizes vertical sonorities over linear trajectories. As a result, the model for tonality, the Spiral Array, does not capture linear patterns. For example, the three adjacent notes centered around the one circled in Figure 8 (F \sharp , F $\sharp\sharp$, G) form an ascending line that is aurally appealing but uses a note that is not a member of the key context.



Figure 8. Linear motion in Op.109 resulting in spelling error in bar 10 (circled).

Errors also occur when the algorithm does not detect a new key context quickly enough. The error in Op.79 (shown in Figure 9) occurred because the c.e. had not yet caught on to the new E minor context beginning in bar 19, thus spelling the D \sharp as an E \flat . However, it adapts quickly and spells the next D \sharp correctly three bars later.

Sudden switches to distant keys are more prevalent in Beethoven's later works. An example is shown in Figure 10, where many of the errors in the Op.109 example occurred. At the double bar lines, the key



Figure 9. Spelling error (circled) in bar 20 of Op.79 because the c.e. had not yet detected the E minor context.

context switches quite abruptly from E major (a key with four sharps) to C major (a key with no accidentals). The cumulative c.e., at this point in time, is entrenched in the E major area and did not detect this switch quickly enough, resulting in the errors shown. The c.e. would be able to track changing keys better if we used a sliding window rather than the cumulative sum from the beginning of the piece. Ideally, the size of the sliding window should be able to adapt dynamically over time with the pace of key changes. The use of sliding windows of adaptive width will be an area for future investigations.

 A musical score for Op. 109, showing a sudden key change. The score is in 2/4 time and features complex rhythmic patterns with triplets and sixteenth notes. The key signature changes abruptly. Several notes are circled in red to indicate spelling errors. A dashed line labeled '8th' indicates an octave shift. The piece consists of two systems of music.

Figure 10. Sudden key change prior to bar 61 in Op.109 resulting in spelling errors (circled).

6. Summary

The paper introduced the problem of assigning pitch spellings to MIDI data. Pitch spelling is an important component of automated transcription, and other applications where MIDI (or some other digital music

format) needs to be transformed to music notation. We introduced the Spiral Array representation and the use of evolving c.e.'s to track changing key contexts. That c.e.'s form trajectories in the interior of the Spiral Array that gravitate toward the appropriate key representations. A pitch spelling algorithm was introduced based on the Spiral Array model.

The algorithm has been incorporated into MuSA, a music visualization software, and tested on two movements from Beethoven's Piano Sonatas Op.79 and Op.109. The proposed algorithm for assigning and correcting pitch spellings was found to be effective in these examples. The algorithm produces a mapping of MIDI to pitch names that is tonally self-consistent. A self-consistent spelling, on rare occasions, may not be the same as that annotated by the composer. One could construct a pathological example in which the algorithm assigns a key of G \flat (index -6) major when the notes should be annotated in F \sharp major (index 6). Even in such circumstances, the assigned pitch names will be consistent with one another and the presumed key.

Apart from mapping MIDI to correct pitch names for music content extraction and analysis, the algorithm can also be a useful tool for detecting and correcting errors in music notation. Further experiments with large benchmark data sets are needed to determine the efficacy of the proposed algorithm and its performance in comparison to other strategies.

GLOSSARY

accidental A symbol denoting the raising or lowering of a note by means of a sharp (\sharp), flat (\flat) or natural (\natural).

chord A *chord* is the simultaneous sounding of two or more notes.

enharmonic equivalence Pitches that are enharmonically equivalent differ from each other in name, but not in any other way where keyboards are concerned. Pairs of notes such as A \sharp and B \flat are treated as one and the same.

equal temperament The practice of tuning a keyboard "out of tune" so that one can play in any key on the same instrument. To demonstrate that this was possible, Bach wrote the "Well-Tempered Clavier."

key A *key* is a compositional principle that implies adherence to the note material of a major or minor scale.

MIDI Musical Instrument Digital Interface. A standard for representing music information in a digital format.

modulation A *modulation* is the change of key in the middle of a composition as a part of its structural organization.

note A *note* is a symbol that represents two properties, pitch and duration.

pitch A *pitch* is a sound of some frequency. High frequency sounds produce a high pitch, and low frequency sounds produce a low pitch.

tonality *Tonality* refers to the underlying principles of tonal music, the system of relations amongst pitches having a “tonic” or central pitch as its most important element.

References

- Cambouropoulos, E. (2001). “Automatic Pitch Spelling: From Numbers to Sharps and Flats,” in Proceedings of the VIII Brazilian Symposium on Computer Music. Fortaleza, Brazil.
- Chew, E. (2002). “The Spiral Array: An Algorithm for Determining Key Boundaries,” Special issue of the LNCS /LNAI on Music and Artificial Intelligence. Heidelberg, Germany: Springer.
- Chew, E. (2001). “Modeling Tonality: Applications to Music Cognition,” in Proceedings of the 23rd Annual Meeting of the Cognitive Science Society. Edinburgh, Scotland.
- Chew, E. (2000). “Towards a Mathematical Model of Tonality,” MIT PhD Dissertation. Cambridge, MA.
- Rowe, R. (2001). “Machine Musicianship,” Cambridge, MA: MIT Press.
- Steedman, M. (1994). “The Well-Tempered Computer,” Philosophical Transactions of the Royal Society. **A:349**. 115–131.
- Temperley, D. (2002). “The Cognition of Basic Musical Structures,” Cambridge, MA: MIT Press.