TREE MODEL OF SYMBOLIC MUSIC FOR TONALITY GUESSING

David Rizo, José M. Iñesta, Pedro J. Ponce de León
Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante, Spain
drizo,inesta,pierre@dlsi.ua.es

ABSTRACT
Most of the western tonal music is based on the concept of tonality or key. It is often desirable to know the tonality of a song stored in a symbolic format (digital scores), both for content based management and musicological studies to name just two applications. The majority of the freely available symbolic music is coded in MIDI format. But, unfortunately many MIDI sequences do not contain the proper key meta-event that should be manually inserted at the beginning of the song. In this work, a polyphonic symbolic music representation that uses a tree model for tonality guessing is proposed. It has been compared to other previous methods available obtaining better success rates and lower performance times.

KEY WORDS
Applications in multimedia, music information retrieval, tonality, cognitive modeling

1 Introduction

In music theory, the tonality or key is defined as the quality by which all the pitches of a composition are heard in relation to a central tone called the keynote or tonic.

The majority of works that model the tonality of a song stored in a symbolic[1]format (digital scores) use linear data structures to represent the sequences of notes [7], [6]. There are other alternatives such as the *spiral array* presented in [1], and under a different approach, a tree representation of monophonic music was introduced in [3] to compare the similarity of musical fragments, obtaining better results than those using linear string representations. In this paper we extend the proposed tree model to represent polyphonic melodies, and use it to find the key of a melodic segment.

The paper is organized as follows: first the monophonic tree representation of music is reviewed, introducing the extension to polyphonic music. After that, how trees are preprocessed is explained before describing the algorithm to calculate the key of the song. We expose the experiments we have performed and give the obtained results. Finally, some conclusions and planned future works are drawn.

[1]In opposition to digital recorded audio.

2 Tree representation for music sequences

A melody has two main dimensions: time (duration) and pitch. In linear representations, both pitches and note durations are coded by explicit symbols, but trees are able to implicitly represent time in their structure, making use of the fact that note durations are multiples of basic time units, mainly in a binary (sometimes ternary) subdivision. This way, trees are less sensitive to the codes used to represent melodies, since only pitch codes are needed to be established and thus there are less degrees of freedom for coding.

In this section we review shortly the tree construction method that was introduced in [3] for representing a monophonic segment of music, defining the terms needed to build the model.

2.1 Tree construction for each measure

Duration in western music notation is designed according to a logarithmic scale: a *whole* note lasts double than a *half* note, that is two times longer than a *quarter* note, etc. (see Fig. 1). The time dimension of music is divided into 'beats', and consecutive 'beats' into bars (measures).

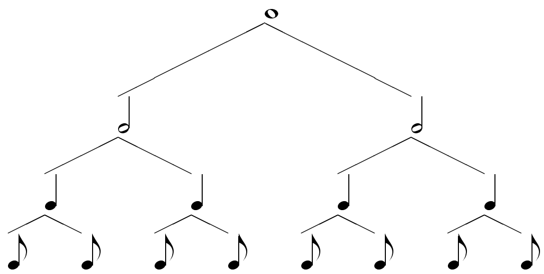

Figure 1. Duration hierarchy for different note figures. From top to bottom: whole (4 beats), half (2 beats), quarter (1 beat), and eighth (1/2 beat) notes.

In our tree model, each melody measure is represented by a tree, τ. Each note or rest will be a leaf node. The left to right ordering of the leaves keeps the time order of the notes in the melody. The level of each leaf in the tree determines the duration of the note it represents, as displayed in figure 1: the root (level 1) represents the duration of the whole measure (a *whole* note), each of the two nodes at level 2 represents the duration of a *half* note. In general, nodes at level i represent the duration of a $1/2^{i-1}$ of a measure.

During the tree construction, internal nodes are created when needed to reach the appropriate leaf level. Initially, only the leaf nodes will contain a label value. Once the tree is built, a bottom-up propagation of these labels is performed to fully label all the nodes. The rules for this propagation will be described in section 2.3.

Labels are codes representing any information related to pitch. In this work labels are the pitch of the note without octave information, also named *folded pitch*, defined by the MIDI note number modulo 12, ranging from 0 to 11. Then, for example, either C_3 or C_4 are considered as C and will be represented as a 0, any C# or D*b* as a 1, and any B as a 11, etc. Rests are coded with a special symbol 's' (for 'silence').

An example of this scheme is presented in Fig. 2. In the tree, the left child of the root has been split into two subtrees to reach level 3, that corresponds to the first note duration (a quarter note lasts a $1/2^2$ of the measure, pitch B coded as **11**). In order to represent the durations of the rest and note G (**7**) (both are 1/8 of the measure), a new subtree is needed for the right child in level 3, providing two new leaves for representing the rest (**s**) and the note G (**7**). The half note C (**0**) onsets at the third beat of the measure, and it is represented in level 2, according to its duration.

It can be seen in figure 2 how the order in time of the notes in the score is preserved when traversing the tree from left to right. Note how onset times and durations are implicitly represented in the tree, compared to the explicit encoding of time needed when using strings. This representation is invariant against changes in duration scaling or different meter representations of the same melody (e.g. 2/2, 4/4, or 8/8).

For a deeper explanation of how to deal with dotted notes, ternary subdivisions, grace notes, and more elaborated examples see the full method in [4].

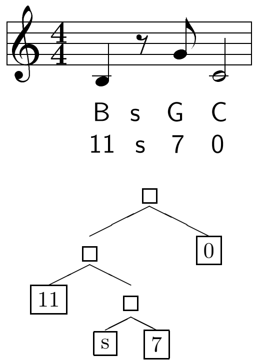

Figure 2. Simple example of tree construction

2.2 Complete melody representation

The method described above is able to represent a single measure as a tree, τ. A measure is the basic unit of rhythm in music, but a melody is composed of a series of M measures. In and previous work[3] it was proposed to build a tree with a root for the whole melody, being each measure sub-tree a child of that root. This can be considered as a forest of sub-trees, but linked to a common root node that represents the whole. Figure 3 displays an example of a simple melody, composed of three measures and how it is represented by a tree composed of three sub-trees, one per measure, rooted to the same parent node. Level 0 will be assigned to this common root.

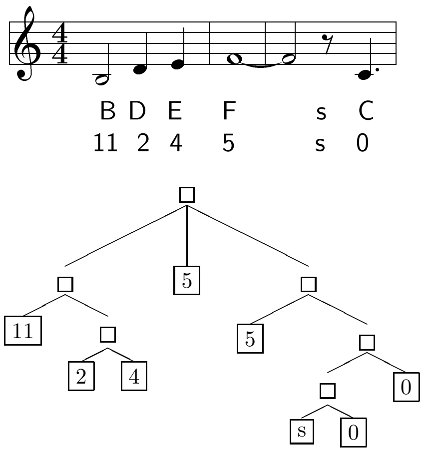

Figure 3. An example of the tree representation of a complete melody. The root of this tree links all the measure sub-trees.

The proposed method to represent polyphonic music is straight forward. All notes are placed in the same tree following the rules of the monophonic music representation. This way two notes with the same onset time will be put in the same node. If a node already exists when a new note is put in the tree, the pitch of this note is added to the node label. If the label in the node is a rest, it is replaced by the note pitch. Figure 4 (center) contains a melody with a chord as an example. Before label propagation (section 2.3), only leaves are labelled.

2.3 Bottom-up propagation of labels

Once the tree is constructed, a label propagation step is performed. The propagation rules are different from those proposed in [3], where the target was similarity search. Now the presence of all the notes is emphasized since every note and chord are tips to find the key of the song segment.

The propagation process is performed recursively in a post-order traversal of the tree. Labels are propagated using set algebra. Let $L(\tau)$ be the label of the root node of the

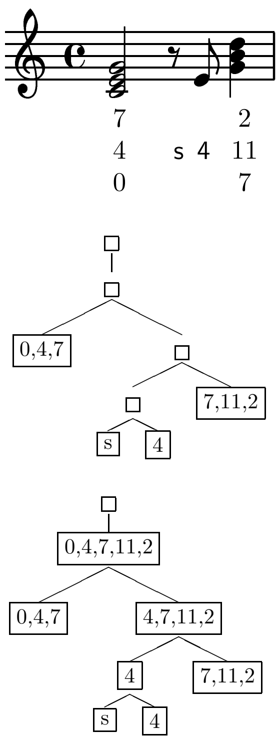

Figure 4. An example of a polyphonic melody (top), its tree representation (center), and the labels for the propagation (bottom)

subtree τ expressed as a set of folded pitches. When the label of the node is a rest, the label set is empty: $L(\tau) = \emptyset$. Then, given a subtree τ with children c_i, the upwards propagation of labels is performed as $L(\tau) = \bigcup_i L(c_i)$.

In figure 4 (bottom) we can see how the eighth note E (folded pitch 4) that shared a parent with the rest is promoted ($\emptyset \cup \{4\} = \{4\}$), and merging this eighth note ($\{4\}$) and the chord next ($\{7{,}11{,}2\}$) results a parent label: $\{4\} \cup \{7, 11, 2\} = \{4, 7, 11, 2\}$. The same procedure is applied for the root of the measure.

3 Key finding algorithm

Each local segment of a melody provides a clue of the possible keys in which it is written in, but the combination of many local clues can reduce the possible keys leaving at the end the correct tonality with a high standard of accuracy.

For example, given a local segment of music with two notes C and G (first bar in the score in figure 5) and want to know which one of the 24 possible keys (12 major, 12 minor) describes this segment the best in terms of tonality. The answer is not a unique key: either C *major*, G *major*, A *minor*, etc. are compatible with those notes. The second measure can be C *major*, A *minor*, but not G *major*. The third measure is probably written in C *major*, and with less probability in A *minor* because although the notes are compatible with those for A *minor*, the present chords are the subdominant, dominant and tonic of C *major*. If we combine the possible keys of the three bars, the most probable answer is C *major*.

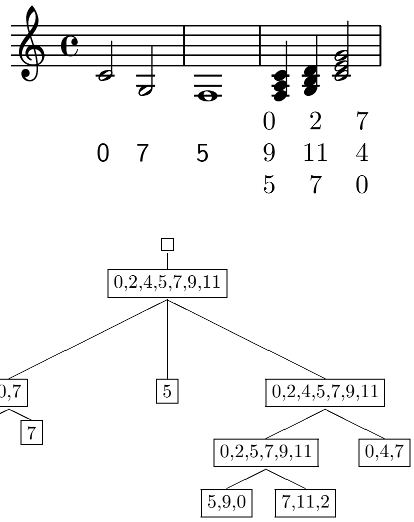

Figure 5. An example of key detection

In our tree model, each node is a local segment that contains one or more folded pitches. In general, several possible keys can be attributed to it. If the possible keys are combined from the leaves to the root following a post-order traversal, finally the root will give us hopefully the most likely key. The tree in figure 5 illustrates this point. The pitch in leftmost node ($\{0\}$) is compatible with C *major*, A *minor*, D*b major*, etc., because that pitch belongs to the diatonic scale of those keys. Its sibling node, labelled with $\{7\}$ can be also C *major*, A *minor*, but not D*b major* because in this tonality, the natural G does not belong to the diatonic scale. If the possible keys for both nodes are combined in their parent node, only C *major* and A *minor* remain valid. Thus, combining node tonality guesses in a post-order way reduces the possibilities.

Computing the candidate keys for each node is performed in two steps:

- First a *rate* is obtained for each of the 24 keys possible applying an algorithm based on rules that will be detailed in section 3.2.
- Then, the keys are ordered decreasingly according to the obtained *rate* resulting in a *rank*, which has the most probable keys first. If two keys have the same rate, they are given the same rank position. The reason for using rank positions instead of rate values is that they make the system more robust against wrong local guesses.

After a rank of keys for each individual node has been created, a post-order combination of these ranks is performed in order to have the final rank at the root of the tree. In this rank the key that appears in the first position is considered the central key of the whole melody. These procedure is summarized in a recursive way in the algorithm 1.

Algorithm 1 Key finding on tree τ

```
if arity(τ) = 0 then
   calculate key ranks for τ root node (see sect. 3.2)
else
   for all child(τ) ∈ children(τ) do
      Algorithm1(child(τ))
   end for
   rank(τ)=combine( ranks(τ, child) ) (see sect. 3.3)
end if
```

3.1 Scales, degrees and chords

3.1.1 Scales

Definition 3.1 specifies the utilized scales represented as a vector indexed by the interval from the tonic note of the key (from 0 to 11), being *M* the major scale, and *m* the minor scale. The values $M[i] \neq 0$ are the degrees in the scale represented in roman numbers. Zero values represent those notes that do not belong to the scale. In the minor scale, m, the natural, harmonic, and melodic scales have been merged.

Definition 3.1 *Diatonic scales*

Major scale M = [I, 0, II, 0, III, IV, 0, V, 0, VI, 0, VII]

Minor scale m = [I, 0, II, III, 0, IV, 0, V, VI, VI, VII, VII]

3.1.2 Degrees

Let a tonality be represented by its key note, represented as a folded pitch, and its mode, *major* or *minor*, defined by the corresponding scale S. Then, given a folded pitch p and a key k, the $degree$ of p is defined as:

$$deg(p, k, S) = S[(((p + 12) - k) \; mod \; 12)] \quad (1)$$

A given scale, S, can be either S = *M* or S = *m*. Given the set of $|P|$ folded pitches $P = \{p_1, p_2, ..., p_{|P|}\}$ in a node, the number of pitches in P that belong to the scale *S* of key *k* is defined as:

$$scaleNotes(P, k, S) = \sum_{i=1}^{|P|} (deg(p_i, k, S) \neq 0) \quad (2)$$

Given the $degree$ for a note in the key, *tonal* and *modal* degrees are considered:

Tonal degrees TD = {I, IV, V}

Modal degrees MD = {III}

Tonal degrees are important to define the keynote, while modal degrees help to distinguish between *major* and *minor* modes.

Given the above definitions, the $tonalDegNotes$ and $modalDegNotes$ functions are defined as:

$$tonalDegNotes(P, k, S) = \sum_{i=1}^{|P|} (deg(p_i, k, S) \in TD) \quad (3)$$

$$modalDegNotes(P, k, S) = \sum_{i=1}^{|P|} (deg(p_i, k, S) \in MD) \quad (4)$$

3.1.3 Chords

Only the diatonic scale triad chords have been considered. The set of notes contained in the label of a node may constitute either a full triad or a partial one. Given the set *P*, $chordNotes(k, c, P)$ is defined as the number of elements in P that belong to a chord c given the key k.

In figure 4 (bottom), the leftmost node in the tree represents the first chord in the score. For it, $P = \{0, 4, 7\}$, for $k = C\ major$ and c =I (the tonic triad of C *Major*). Therefore $chordNotes(k, c, P)$=3 because it contains the three pitches of this chord. If $k = A\ minor$ is considered and c =I again (*A minor* tonic triad composed by the pitches *A*, *C* and *E*), the result would be 2 because only the pitches *C* and *E* are found in the chord.

3.2 Node key rating

Given the previous definitions, the rules in table 1 compute the rate value for each key according to the set of pitches in a node. These rates (see table 2), have been established empirically after an exhaustive search over the parameter space.

This scheme scores triad chords that clearly belong to a key the highest. Then it gives lower values both to two note chords and single notes that belong to the key.

Constant	Rate
FULL_TRIADS_I_V	16
FULL_TRIADS	15
2NOTES_TRIADS_I_V	9
2NOTES_TRIADS	8
NOTES_CHORDS_I_V	10
2NOTES_CHORDS	9
TONAL_DEGREES	4
MODAL_DEGREES	3
SCALE_NOTES	2

Table 2. Rates values for constants in table 1

Rule	$\|P\|$	Condition	Rate
1	3	$chordNotes(c) = 3\ where\ c \in \{\mathrm{I}, \mathrm{V}\}$	*FULL_TRIADS_I_V*
2	3	$chordNotes(c) = 3\ where\ c \in \{\mathrm{II}, \mathrm{III}, \mathrm{IV}, \mathrm{VI}, \mathrm{VII}\}$	*FULL_TRIADS*
3	3	$chordNotes(c) = 2\ where\ c \in \{\mathrm{I}, \mathrm{V}\}$	*2NOTES_TRIADS_I_V*
4	3	$chordNotes(c) = 2\ where\ c \in \{\mathrm{II}, \mathrm{III}, \mathrm{IV}, \mathrm{VI}, \mathrm{VII}\}$	*2NOTES_TRIADS*
5	2	$chordNotes(c) = 2\ where\ c \in \{\mathrm{I}, \mathrm{V}\}$	*2NOTES_CHORDS_I_V*
6	2	$chordNotes(c) = 2\ where\ c \in \{\mathrm{II}, \mathrm{III}, \mathrm{IV}, \mathrm{VI}, \mathrm{VII}\}$	*2NOTES_CHORDS*
7	$\|P\| - SN > 2$	$tonalDegNotes(P, k, S) > 0$	*TONAL_DEGREES*
8	$\|P\| - SN > 2$	$modalDegNotes(P, k, S) > 0$	*MODAL_DEGREES*
9	$\|P\| - SN > 2$	$scaleNotes(P, k, S) > 0$	*SCALE_NOTES*

Table 1. Key k rating, for the node pitches P. The rules are checked and applied if the conditions are met in precedence order from rules 1 to 9, firing only the first matched rule. SN stands for $scaleNotes(P, k, S)$

3.3 Subtree key bottom-up combination

Once the ranks for the children nodes and the parent node have been calculated, they must be combined to replace all the key ranks in the parent node. This operation is performed in two steps. First the rate values for the parent node are recomputed, and then a new sort of the tonalities is performed.

Given a parent tree node τ, with children $c_1, c_2, ..., c_{arity(\tau)}$, the new rate for each key k is calculated as:

$$rate(\tau, k) = rank(\tau, k) + \sum_{i=1}^{arity(\tau)} rank(c_i, k) \qquad (5)$$

The function $rank(\tau, k)$ returns the position of tonality k in the rank for the root node of τ.

4 Experiments and results

In order to evaluate our algorithm, 212 MIDI files of classical pieces have been collected, including Cimarosa, Albinoni, Bach, Beethoven, Chopin, Dvorak among others[2]. To avoid key changes as far as possible, the first 8 measures for each song have been extracted.

We have compared our method with two freely available systems. One is the *key* program of *Melisma*[3]. This system implements three different algorithms that can be selected: *CBMS* ([5]), a *Bayesian* key-finding algorithm [6] and the *Krumhansl-Schmuckler* (KS) algorithm ([2]). The other is the program *key* of *Humdrum* (HUM)[4], which also implements the *Krumhansl-Schmuckler* method with the parameters that the authors established in [**?**].

The *key* program of *Melisma* returns a list of keys ordered in time. The central key is calculated as the most repeated one.

[2]The database is available for research purposes under request to the authors

[3]Version 2003 implemented in http://www.link.cs.cmu.edu/music-analysis/

[4]http://dactyl.som.ohio-state.edu/Humdrum/

Relation to correct key	Points
Same	1
Perfect fifth	0.5
Relative major/minor	0.3
Parallel major/minor	0.2

Table 3. MIREX 2005 Key finding scorings

To compare the results, we have followed the evaluation process proposed for the *Audio and Symbolic Key Finding* topic of the 2nd Annual Music Information Retrieval Evaluation eXchange (MIREX 2005) [5], as detailed in table 3.

The success rate of an algorithm is obtained as the achieved points averaged for all the songs in the corpus.

The *Melisma* system is built in ANSI C, our system uses the Java 1.4.2-38 virtual machine, that is even slower than the native code generated from C. All the experiments have been run in a Apple PowerBook, using a PowerPC G4 1.33 Ghz processor with 512 Mb of RAM memory.

The plot in figure 6 shows the results of average scorings and total times for our algorithm (*Trees*), each one of the three methods implemented in *Melisma*, and the algorithm in *Humdrum*. The best rates are those giving a value closer to 1. The *Trees* algorithm performs the best and requires around eight times less computation time than the others.

5 Conclusions and future work

In this work we have presented a polyphonic music tree representation that has proved to be a simple and adequate representation for finding the key of a song. The success rates were slightly better than those of *Melisma* and *Humdrum* systems but the computing times are much smaller.

[5]http://www.music-ir.org/mirexwiki/index.php/MIREX_2005

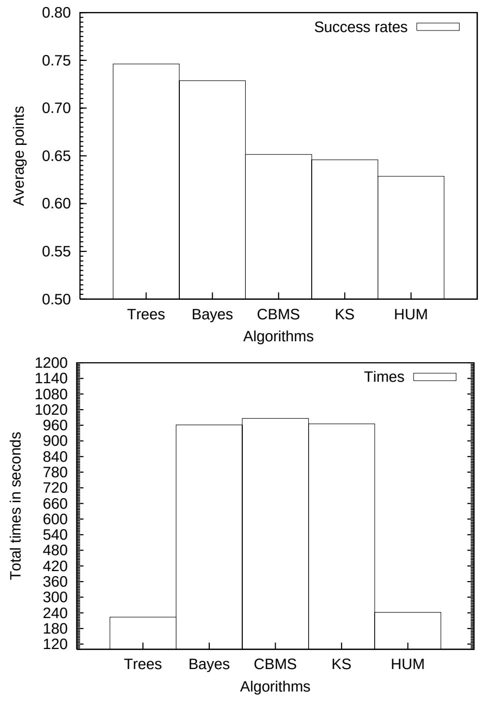

Figure 6. Average points and total times

The proposed method utilize very little harmonic information, but nevertheless a good key identification has been achieved. The system can be improved by using of a more powerful harmonic model. Also the rates for scoring, now empirically obtained, could be automatically learned from a given training set, providing more flexibility and robustness to the method.

We are also working in the key change finding inside a given song obtaining some promising results.

6 Acknowledgments

This work was supported by the projects Spanish CICYT TIC2003–08496–C04, partially supported by EU ERDF, and Generalitat Valenciana GV043-541.

References

[1] Elaine Chew. Modeling Tonality: Applications to Music Cognition. In Johanna D. Moore and Keith Stenning, editors, *Proceedings of the 23rd Annual Meeting of the Cognitive Science Society*, pages 206–211, Edinburgh, Scotland, UK, August 1-4 2001. Lawrence Erlbaum Assoc. Pub, Mahwah, NJ/London.

[2] C. Krumhansl. *Cognitive Foundations of Musical Pitch*. Oxford University Press, New York, NY, USA, 1990.

[3] D. Rizo, F. Moreno-Seco, and J.M. Iñesta. Tree-structured representation of musical information. *Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence*, 2652:838–846, 2003.

[4] David Rizo and José M. Iñesta. Tree-structured representation of melodies for comparison and retrieval. In *Proc. of the* 2^{nd} *Int. Conf. on Pattern Recognition in Information Systems, PRIS 2002*, pages 140–155, Alicante, Spain, 2002.

[5] D. Temperley. *The Cognition of Basic Musical Structure*. MIT Press, 2001.

[6] D. Temperley. A bayesian approach to key-finding. *Lecture Notes in Computer Science*, 2445:195–206, 2002.

[7] Y. Zhu and M. Kankanhalli. Key-based melody segmentation for popular song. *17th International Conference on Pattern Recognition (ICPR'04)*, 3:862–865, 2004.