AUTOMATED PARTITIONING OF TONAL MUSIC

BRYAN PARDO

WILLIAM P. BIRMINGHAM

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE DEPARTMENT
THE UNIVERSITY OF MICHIGAN

20 AUGUST, 1999

TECHNICAL REPORT

CSE-TR-396-99

Abstract

The majority of research related to automated analysis of music presupposes human partitioning of the input into segments corresponding to significant harmonic or melodic chunks. In this paper, we analyze the difficulty of the partitioning problem in a formal manner. We then describe HarmAn, a system that partitions tonal music into harmonically significant segments corresponding to single chords and labels these segments with the proper chord labels. Input to the system consists of standard MIDI files and the output is both an annotated piano-roll style display and a text file with partitioning and chord-name information. Chord labels for segments are determined through template matching in the space of pitch-class with conflict resolution between equal scoring templates resolved through simple default preference rules. Our system's results are compared with the results described in papers by Winograd (Winograd 1968), Maxwell (Maxwell 1992), and Temperley and Sleator (Temperley and Sleator 1999).

1 Introduction

Since the 18th century, the vast majority of widely recognized pieces of music written in both the art and popular traditions of Western Europe and North America have used tonality and tonal chord structures based on triadic harmonies as a basic structural feature of the music. Central to the understanding of any tonal or tonal-based piece of music is an understanding of what harmonies are used in the piece. To determine which harmonies are used, one must be able to temporally partition the music into segments that divide the music at points where the harmonies change and label the segments appropriately. We call this the partitioning problem.

Previous work in the area of automated harmonic analysis of music (Winograd 1968; Smoliar 1980; Maxwell 1992; Widmer 1992; Smaill, Wiggins et al. 1993), with the notable exception of recent work by Temperley and Sleator (Temperley and Sleator 1999), has either avoided the issue of generating a good partitioning by taking partitioned input or has been unclear in how the issue is resolved. Further we are unaware of any published work that clearly lays out a complexity analysis of partitioning and labeling a piece of tonal music. We address this by creating a theoretical framework that allows the problem to be formally analyzed.

Finding a good partitioning requires a metric for determining the "goodness" of a partitioning. We describe a concise template matching algorithm, related to the work of both Ulrich (Ulrich 1977) and Wakefield (Wakefield 1999; Wakefield and Pardo 1999) that quickly labels a partitioning's segments and generates a score for the partitioning. Our approach decouples labeling a single partitioning from finding the best one. This allows the use of a variety of generic search methods to find a good partitioning.

We show that a piece of music with n notes may have up to 2^{2n-2} possible partitionings and then fully describe a method to arrive at a good partitioning of the music by searching only $O(n)$ partitionings. The combination of our template matching and method for search through the space of partitioning produces results that compare well to systems comprised of hundreds of production rules.

The following section describes the theoretical framework for harmonic analysis of tonal music used in the remainder of the paper.

2 Notation and Terminology

2.1 The Note

Let a *note*, *n,* be defined as a 4-tuple of the form <*start, end, pitch class, octave*> where *n*:

> *start* is a real number giving the number of seconds between the start of note *n* and the start of the first note in the piece.
>
> *end* is a real number giving the number of seconds between the end of note *n* and the start of the first note in the piece.
>
> *pitch_class* is an integer from 0 through 11 representing the pitch class of note *n*
>
> *octave* is an integer from 0 through 11 representing the octave in which note *n* occurs.

The first "C" in Figure 1 is an example of a note and is represented as the tuple <0, 1, 0, 4>.

When referring to an element in a note tuple the field is referred to by name and the identity of the note is denoted by a subscript. For example, the pitch class of note *n* is referred to as $pitch\_class_n$. If *n* is the first "C" in Figure 1, then $pitch\_class_n = 0$.

Rests are not explicitly represented in the manner of notes. A rest is a segment of time where no notes sound, and is defined as such in Section 2.6.

2.2 The Piece of Music

A *piece of music*, *M*, is defined as a set of notes.

Identical notes are allowed by this definition of a piece. For example, it may be that there is a unison note between two voices. In this case, there would be two identical notes in the set *M*.

M_x denotes a particular performance of a piece of music, where *x* is the label for the performance.

Consider the Beethoven excerpt in Figure 1. Assume that the tempo is one quarter note per second. We will use this tempo as the default for all examples relating to the Beethoven excerpt. It is assumed, unless otherwise stated, that the tempo remains constant throughout the piece. Let the performance $M_{beethoven1}$ be defined as the first measure of the piece when played at one quarter note per second.

$M_{beethoven1}$ = {<0, 1, 0, 4> , <0, 0.25, 9, 3> , <0, 1, 9, 2>, <0.25, 0.5, 3, 3> ,
<0.5 ,0.75, 9, 3> , <0.75, 1, 3, 3> , <1, 2, 11, 3> , <1, 2, 1, 3> ,
<1, 1.25, 7, 3> , <1.25, 1.5, 3, 3> , <1.5, 1.75, 7, 3> , <1.75, 2, 3, 3>}

2.3 Time and Meter

In this paper, beats are often used in order to refer to the written notation of an example. Note, however, that there is no explicit reference to beats or metrical information in the definition of a note. Nor is there any explicit reference to metrical information in any structure based on notes. This allows music lacking a basic metrical pulse to be represented and manipulated as easily as more rhythmic music.

All timing information is defined in terms of the number of seconds since start of the earliest sounding note of the piece. The minimum value for time is 0 and the maximum value is the end of the final note to sound in the piece. Since real times are used, the definition of a piece of music is tied to a particular performance of that piece. Different performances may result in timing variations that will change the definition of the piece.

Time is assumed to be continuous and represented by a real number. The choice of continuous rather than discrete time was made to avoid basing the theoretical treatment on an arbitrary underlying quantization. The closest we come to a quantum of time in our treatment of music is the *minimal segment* (defined in Section 2.6), which has a duration that varies with the tempo and density of notes.

2.4 The State of the Piece

The *state* of the music at time *t*, $State_t$, is the subset of notes from a piece of music, *M*, that is sounding at time *t*.

$$State_t = \forall (n \in M) \mid [(start_n < t) \wedge (end_n > t)]$$

Taking $M_{beethoven1}$ as the piece of music, the state at time 0.3 is a set of four notes.

$State_{0.3}$ = {<0, 1, 0, 4> , <0, 1, 9, 2>, <0.25, 0.5, 3, 3>}

HarmAn, the music analysis program described later in this paper, uses only the pitch class of the notes sounding at time *t*. For this reason, HarmAn represents state as a 12 element array indexed by pitch class number (0 through 11). Each element, *i*, gives the count of notes of pitch class *i* that are sounding at time *t*. In this representation, $State_{0.3}$ is represented by the following array.

$State_{0.3}$ = [1,0,0,1,0,0,0,0,0,1,0,0]

2.5 Partition Points and Partitionings

Each time where the state changes is a *partition point.* This occurs when a note either starts or ends. From this, it is clear that each partition point, p, corresponds to the start or end of one or more notes.

The *set of all partition points*, P_{all}, may be derived from the notes in M by finding all $start_n$ and end_n and removing any duplicates. Since each note has a start time and an end time, the size of P_{all} can be no more than twice the size of M. This means that P_{all} is finite and countable as long as M is finite and countable (a reasonable assumption for a piece of music).

$$2|M| \geq |P_{all}|$$

Given $M_{beethoven1}$ as the piece of music, the set of partition points, $P_{all,}$ is

$$P_{all} = \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2\}$$

Each partition point, p_i, in P_{all} represents a unique time. An ordering can be imposed on P_{all} through sorting its elements by value. Let the partition point with the earliest time be p_1 and the point with the latest time be $p_{|Pall|}$. The ordering relation for P_{all} is expressed as follows.

$$\forall(i, j) : (p_i < p_j) \text{ iff } (i < j)$$

If $P_{all} = \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2\}$, then p_1 is 0, p_3 is 0.5 and $p_{|Pall|}$ is 2.

Figure 1 shows a passage from Beethoven's Sonata Pathetique. Figure 2 represents the same passage in a piano-roll style notation, where each note is represented by a line. Vertical position represents the height of the note, length represents the duration and horizontal position represents time since the beginning of the piece of music. A vertical line is placed at each partition point.

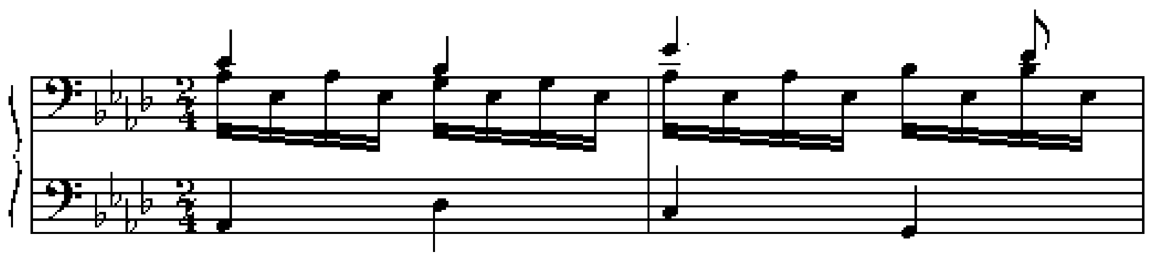


Figure 1: Beethoven, Sonata Pathetique, Op 13, Second Movement, mm 1-2

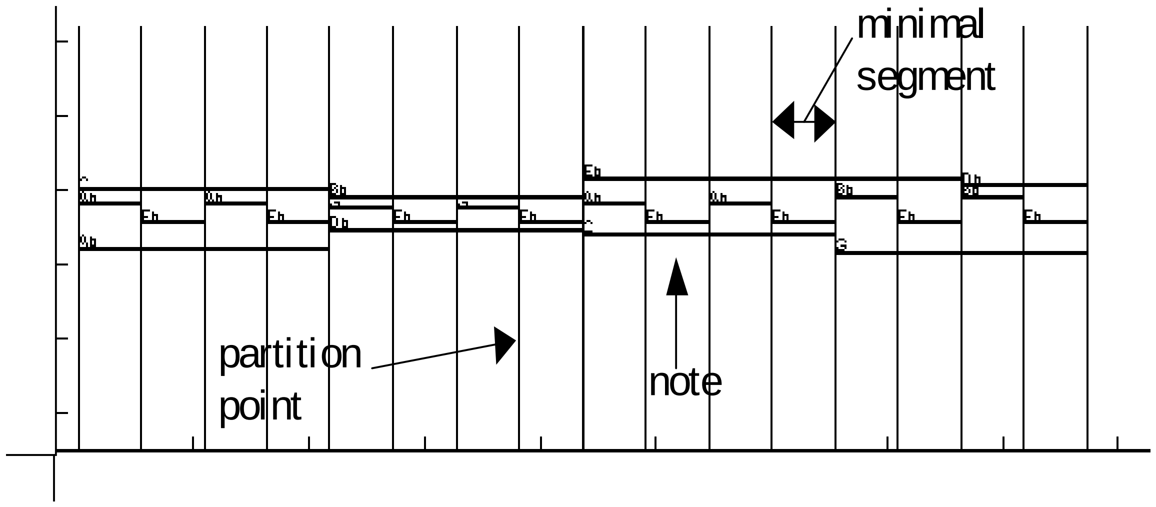


Figure 2: Beethoven, Sonata Pathetique, Op 13, Second Movement, mm 1-2

Although the example in Figure 2 has partition points occurring every quarter of a second, this is strictly a result of the even tempo of the performance. If the performance were to vary in tempo, then the length of the minimal segment would vary as well. This can also occur in a piece with an even tempo, but with notes of varying length. Consider the Debussy excerpt in Figure 3. Assume a constant tempo of one quarter note per second and that each note begins the exact instant the previous note ends. In this case,

$$P_{all} = \{0, 1.5, 1.75, 2, 2.5, 2.6\underline{66}, 2.8\underline{33}, 3, 3.5, 4, 6.5, 6.75, 7, 8\}.$$

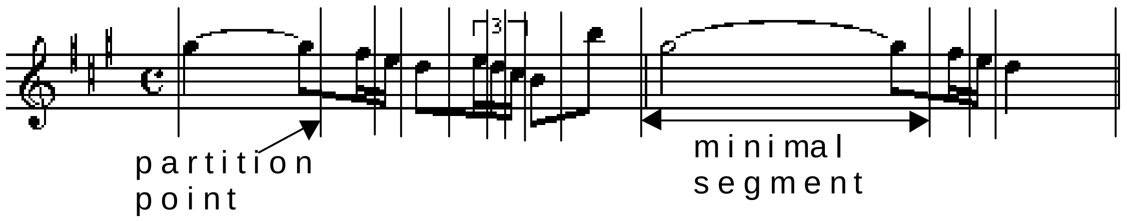


Figure 3: Debussy, The Little Shepherd, mm 1-2

Note that in the Debussy example, partition points are spaced anywhere from roughly 0.166 to 2.5 seconds apart.

A *partitioning*, *P*, of a piece of music is a subset of the set of partition points, P_{all}, including elements p_1 and $p_{|Pall|}$.

Since the elements of P_{all} are sorted in temporal order, a partitioning may be represented by a binary number where bit *i* indicates whether partition point *i* should be used to partition two segments. We assign "0" (for "not in partitioning") or "1" (for "in partitioning") to each bit and the resulting binary number uniquely identifies a partitioning of the music.

Every partitioning includes the first and last elements of P_{all}. The first and last elements of P_{all} will always have their bits set to "1" and the bits that uniquely identify a partitioning are those for partition points p_2 through $p_{|Pall|-1}$. Thus, any partitioning can be uniquely identified by a number with $|P_{all}| - 2$ bits.

There are nine partition points in the first measure of the Beethoven example. The set of all partition points for this measure, P_{all}, can thus be represented by the seven digit binary number "1111111." The digits in this number represent partition points p_2 through $p_{|Pall|-1}$.

Two example partitionings of the first measure of the Beethoven example are shown in Figure 4. Each vertical line corresponds to a partition point. Thick lines correspond to "1"s and represent the partition points that are in both P_{all}, the set of all partition points, and P, the partitioning. Thin lines correspond to "0"s and represent members of P_{all} that are not in P. Partitioning 0001000 in Figure 4 divides the measure into two equal segments. Partitioning 0001011 divides the measure into four segments of varying duration.

A *good partitioning* is one consisting of only the elements in P_{all} that correspond to harmonically significant changes in the state of the music. Partitioning 0001000 in Figure 4 is a good partitioning. The second partitioning in Figure 4 is not a good partitioning.

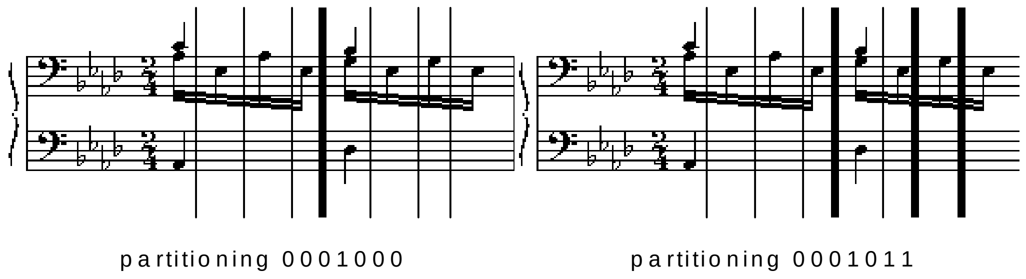


Figure 4: Two partitionings of Beethoven, Sonata Pathetique, Op 13, 2nd Mvmt., measure 1

Points of *harmonically significant* change are those partition points where the chord name that a human would assign to the current state changes. The problem of finding a *good partitioning* for a piece of music is determining which partition points are harmonically significant.

2.6 Segments and Segmentations

A *segment*, *s*, is the interval between partition points, p_i and p_j. Define *s* as a duple, <*start, end*> where (*start, end*) $\in P_{all}$ and *start* < *end*.

A *rest* is a segment in which no notes sound.

A *minimal segment* is a segment between two sequential partition points in P_{all}, p_i and p_{i+1}. Figure 2 identifies a minimal segment, as does Figure 3. The state does not change for the duration of a minimal segment. The duration of a minimal segment depends only on how long the state remains constant and may vary between minimal segments in the same piece of music. Figure 3 is an example of this.

All *rests* are *minimal segments*.

Any segment between partition points p_i and p_j, where $j > i + 1$ incorporates at least one change of state is not minimal and can be decomposed into minimal segments.

The *length of a segment* is the number of minimal segments into which the segment may be decomposed. This can be derived from the number of partition points encompassed by the segment. If a segment *s* is defined by $<p_i, p_j>$, the length of *s* is given by $j - i + 1$.

A partitioning, *P*, defines a set of segments, *S*, called a *segmentation*. If the elements of *P* are ordered by increasing time, then each segment in *S* is defined as a duple $<p_i, p_{i+1}>$. The size of set *S* is $|P| - 1$.

Given a constant tempo of one quarter note per second, partitioning 0001011 from Figure 4 contains the set of partition points {0, 1, 1.5, 1.75, 2}. This defines a segmentation {<0,1>, <1,1.5>, <1.5,1.75>, <1.75,2>}.

The *set of all possible segments*, S_{all}, for a piece of music, *M*, contains every duple $<p_i, p_j>$ where $i < j$, drawn from the set of all partition points, P_{all}. The size of this set is $|P_{all}|$ *choose 2*, or $|P_{all}|(|P_{all}| - 1) / 2$.

$$|S_{all}| = |P_{all}| \textit{ choose } 2 = |P_{all}|! / 2(|P_{all}| - 2)! = |P_{all}|(|P_{all}| - 1) / 2 \approx |P_{all}|^2/2$$

With $M_{beethoven1}$, S_{all} is shown in Section 6, Table 4.

3 Complexity Analysis of the Partitioning Problem

Given a piece of music, *M*, and its set of partition points, P_{all}, the partitioning problem is that of finding a good partitioning, $P_{good} \subseteq P_{all}$, that contains only the partition points corresponding to harmonically significant changes in the state of the music.

Since any piece of music has no more than $2|M|$ partition points, each represented by a binary digit, the maximum number of ways to partition a piece of music is bounded by $2^{2|M|}$. This upper bound may be tightened by using the fact that p_1 and $p_{|Pall|}$ correspond to the beginning and end of the piece and are always members of every partitioning. Thus, any partitioning can be uniquely represented by a binary number of at most $2|M| - 2$ digits. This gives a maximal number of $2^{2|M| - 2}$ possible partitionings for any piece of music.

$$2^{2|M| - 2} \geq 2^{|Pall| - 2} \text{ ways to partition M}$$

A *monophonic* piece of music is one where no two notes sound at the same time This can be expressed in terms of the states of a piece, M, at time t.

$$\text{IsMonophonic}(M) \Leftrightarrow \forall t : |State_t| < 2$$

An unaccompanied melody is an example of monophonic music. Even a very simple melody such as “Happy Birthday” contains 25 notes, giving a maximum 2^{48} ways to partition the melody. This number is greatly reduced if one assumes that note i begins at the exact moment note i-1 ends. This brings the number of partition points to $|M|+1$ and the number of ways to partition the piece to $2^{|M|-1}$, or 2^{24} in the case of “Happy Birthday.”

The value of $2^{|M|-1}$ is a lower bound on the number of possible partitionings for any monophonic melody. This can be seen by noting that for a monophonic piece of music the minimum number of states is equivalent to the number of notes. If one separates two notes by a rest, then an additional state is introduced and the number of partition points is increased. If two notes sound at the same time, then the music is no longer monophonic. As can be seen from this analysis, there are a prohibitively large number of ways to partition even a short monophonic melody.

Not all music is monophonic and the number of different ways to partition a non-monophonic piece of music can be significantly lower than even $2^{|M|-1}$. This is because number of possible ways to partition a piece is not determined by the number of notes, but by the number of partition points, $|P_{all}|$. When many notes start or end concurrently, the number of partition points is reduced. An example is a piece consisting of block chords, with the extreme case being one in which all notes in the piece start and end at the same time. Such a situation is rare. A more typical example is that of the Beethoven fragment in Figure 2. The fragment has 24 notes, but only 17 partition points, due to concurrent start and end times for multiple notes. This results in many fewer ways to partition the fragment but the number of partitionings is still 2^{15}.

Out of the $2^{|Pall| - 2}$ ways into which a piece M may be partitioned, relatively few partitionings will correspond to how a human analyst would partition the piece on basis of the harmony. Presuming that a good partitioning, P_{good}, is somehow found, the problem of labeling the segments defined by the partitioning remains. Given that there are c different labels that may be used and that the chosen partitioning has $|P_{good}|$-1 segments, then there are roughly $c^{|Pgood|-1}$ ways of labeling the partitioning. Taken in light of the number of possible partitionings, one can see that an already enormous set of possible variations becomes even more immense when one takes into account labeling the segments in each partitioning.

The size of this set of variations can be reduced by noting two different partitionings may contain one or more segments in common. The *constraint of locality* states that each segment can be labeled in isolation, without reference to other segments. When searching through the partitionings of a piece to find a good partitioning, the same segment will be encountered more than once. If the constraint of locality holds, context does not matter and there is no need to re-calculate the label of a segment the second time it is encountered. This can be achieved by labeling segments with jazz-style chord symbols, such as “A minor 7,” that do not reference their function within the key.

Given the constraint of locality, one need label only the *set of all possible segments*, S_{all}, for a piece of music. All segments associated with any partitioning are subsets of S_{all} and thus no additional segments must be labeled once S_{all} is labeled.

Let c be the number of possible chord labels. Given that $|S_{all}| = |P_{all}|(|P_{all}|-1) / 2$, labeling all possible segments involves making no more than $c|P_{all}|(|P_{all}|-1) / 2$ comparisons.

Once all possible segments in a piece are labeled and placed in a lookup table, the size of the space to be searched in the course of generating a labeled good partitioning can again be considered to be $2^{|Pall|-2}$. We use this value as the size of the search space for the problem of finding a good partitioning of a piece of music.

4 Templates for Segment Labeling

In order to label segments, there must be a way to evaluate the significance of the notes in a segment so as to be able to categorize it. This section discusses the general framework we use to approach labeling sets of notes.

The music we are concerned with is based on a set of 12 pitch classes in the chromatic scale. The common labels for the pitch classes, along with their numeric equivalents, are given in Table 1.

Table 1: Pitch Class Number and Name Correspondences

C	C# / D♭	D	D# / E♭	E	F	F# / G♭	G	G# / A♭	A	A# / B♭	B
0	1	2	3	4	5	6	7	8	9	10	11

In the audio domain, a *pitch class* represents a set of harmonic sounds whose fundamental frequencies are related by a power of 2. An example is the set of “A”s . Assume a sound with a fundamental frequency of 440 Hz is an “A.” All harmonic sounds whose fundamental is $(2^n)*440$, where n is an integer, are also in the pitch class “A.” Thus, sounds at 110, 220, 440, 880 and 1660 Hz are all members of the pitch class “A.”

Each time the frequency of a pitch doubles, the *octave* increases by one. In equal-tempered tuning (the most common tuning in use today), the pitch classes divide an octave into 12 steps, which are equally spaced in the $\log_2$ of the frequency. Once the frequency has doubled, the pitch class label wraps around to the name used one octave below. This repeating 12-step structure is called the *chromatic scale*.

The chromatic scale and its associated 12 pitch classes form the basic set of items used to generate the structures associated with most Western music. Using the integer representations of the pitch classes and modulo 12 arithmetic, structures such as chords and scales, can be represented as *n*-tuples representing positive displacements in the space of pitch classes in relation to a root pitch class. These tuples form templates that are useful for describing musical structures and are related to those used in atonal set theory (Forte 1973), the chromagram (Wakefield 1999; Wakefield and Pardo 1999), and the work of Ulrich (Ulrich 1977).

An example of the template representations is the following. Given a root (pitch) class, *r*, the tuple <0,4,7> represents the pitch class relations to *r* embodied in a major triad. Letting $r = 2$, this results a chord given by mod12(r +0, r+4, r+7) = {2,6,9}. Looking at Table 1, it is easy to verify that these numbers correspond to {D, F#,A}, the pitch classes in the D major triad. Examples of some of the more common tonal structures and their template representations are given in Table 2. These templates are central to the approach we take to chord labeling in the work described in this paper.

Table 2: Common Tonal Structure Representations

NAME OF TONAL STRUCTURE	EXAMPLES OF WRITTEN NOTATION ASSUMING THE ROOT NOTE IS "C"	TEMPLATE REPRESENTATION
major triad	C, C Maj, C major, C:I	mod12(r +0, r+4, r+7)
minor triad	C min, C minor, c:i	mod12(r +0, r+3, r+7)
augmented triad	C+, C augmented, c:I+	mod12(r +0, r+4, r+8)
diminished triad	C dim, C diminished, c:i^o	mod12(r +0, r+3, r+6)
major-minor (dominant) 7th chord	C7, C dom 7, C dominant, F:V7	mod12(r +0, r+4, r+7, r+10)
major scale	C major scale	mod12(r +0, r+2, r+4, r+5, r+7, r+9, r+11)

5 HarmAn

Using the theoretical framework described in the previous sections, we have developed HarmAn, a system that partitions tonal music into harmonically significant segments and labels these segments with the proper chord labels. Input to HarmAn consists of standard MIDI files and the output is both an annotated piano-roll style display and a text file with partitioning and chord-name information. Chord labels for segments are determined through template matching in the space of pitch-class with conflict resolution between equal scoring templates resolved through simple default preference rules.

HarmAn searches the space of $2^{|Pall| - 2}$ partitionings of the music through the examination of only $|P_{all}|$–1 partitionings and returns results that compare well to both to the results generated by other automated systems for harmonic analysis and the results returned by a human expert.

Figure 5 shows the three main processing steps HarmAn takes in generating an analysis of the harmonies of a piece of music. In Step 1, MIDI2PallSm, the system generates the set of all partition points P_{all}, and the set of all minimal segments S_m. In Step 2, HarmAn searches through possible partitionings, and returns the best one found, P_{good}, and the associated labeled segmentation, S_{good}. Step 3 is where HarmAn outputs the results of the harmonic analysis to both the screen and an output file.

GIVEN: a piece of music represented as an input stream of MIDI note events.
Each note event is of the form <*deltaTime, keyNumber, velocity, eventType* >

RETURN: P_{good}, an ordered set of partition points representing a good partitioning of the music

S_{good} an ordered set of segments associated with P_{all}
Each $s \in S_{good}$ is of the form < p_{i-1}, p_i, *label, score, state*>, begins at p_{i-1} and ends at p_i
where p_{i-1} ,p_i are partition points i-1 and i in P_{good}

METHOD

```
1.    [$P_{all}$,$S_m$] = MIDI2PallSm(MIDI_STREAM)

2.    [$P_{good}$, $S_{good}$] = FindGoodPartitioning($P_{all}$,$S_m$)

3.    OutputResults($P_{good}$, $S_{good}$)
```

Figure 5: HarmAn

The remainder of Section 5 describes and analyzes Steps 1 and 2 (functions “MIDI2PallSm” and “FindGoodPartitioning”) in detail. Note that “FindGoodPartitioning” calls “LabelAndScoreSegment ” and that “LabelAndScoreSegment ” calls “FindBestRoot.” Each of these lower-level functions is also described in detail. Step 3 is not described, as it contains code for display and file creation, and is not central to HarmAn’s processing.

5.1 Generating the Initial Partitioning: MIDI2PallSm

To find a good partitioning for a piece of music, *M*, one must first find the set of all partition points. In the case of HarmAn, *M* is represented by a standard MIDI file. The system reads note events from the file and generates the set of all partition points, P_{all} and a set, S_m, containing the state of the piece in each minimal segment. Pseudo code for this is given in Figure 6. Note that the pseudo code assumes “well formed” MIDI where there is each “note on” is followed by a corresponding “note off.”

```
GIVEN:      an input stream of MIDI note events.
                Each note event is of the form <deltaTime, keyNumber, velocity, eventType >

RETURN:     $P_{all}$ ,   an ordered set of all partition points.
                Each $p_i \in P_{all}$ is a real number corresponding to the time of a "note on" or "note off" event

            $S_m$     an ordered set of minimal segments associated with $P_{all}$
                Each $s_{<p_{i-1}, p_i>} \in S_m$ is of the form < $p_{i-1}$, $p_i$, label, score, state>, begins at $p_{i-1}$ and ends at $p_i$

METHOD

1.      integer array     state[0,11] := 0     % the count of notes of pitch class 0 through 11 currently sounding
2.      real              $p_i$ , $p_{i-1}$ , time := 0,
3.      integer   keyNumber, velocity, i

4.      $P_{all}$ := {$p_{i-1}$}

5.      [deltaTime,keyNumber, eventType] := getNoteEvent(MIDI_STREAM)

6.      WHILE eventType = "note on" OR eventType = "note off"

7.          time := time + deltaTime
8.          $p_i$ := time

9.          pitchClass = mod12(keyNumber)

10.         IF $p_i$ <> $p_{i-1}$
11.             $P_{all} := P_{all} \cup \{ p_i \}$
12.             $s_{<p_{i-1}, p_i>}$ = < $p_{i-1}$, $p_i$, "no label", 0, state>
13.             $S_m := S_m \cup \{ s_{<p_{i-1}, p_i>} \}$
14.         END

15.         IF eventType = "note on" THEN state[pitchClass] := state[pitchClass] + 1 END
16.         IF eventType = "note off" THEN state[pitchClass] := state[pitchClass] – 1 END

17.         $p_{i-1}$ := $p_i$

18.         [deltaTime,keyNumber,velocity eventType] := getNoteEvent(MIDI_STREAM)
19.     END
```

Figure 6: MIDI2PallSm

For the purpose of this paper, a MIDI note event is defined to consist of a *deltaTime*, which is the absolute time since the previous note event, a *keyNumber*, which is a value from 0 to 127 specifying the key on a piano style keyboard which was struck, a *velocity*, which is a value from 0 to 127 defining the velocity (usually associated with volume) with which the key was struck, and an *eventType*, which is either "note on" or "note off."

The lowest possible "C" on a MIDI keyboard is represented by 0, "C sharp" is 1 and so on Every 12 keys, the octave increases by one. Thus, pitch class (see Table 1 for the correspondence of pitch class and number). can be derived from *keyNumber* by taking mod12(*keyNumber*).

MIDI2PallSm begins by initializing variables in Steps 1 through 4. The variable *state* is a 12 element array indexed by pitch class number (0 through 11) that represents the state of the music. Each element, *i*, gives the count of notes of pitch class *i* at sounding time *t*.

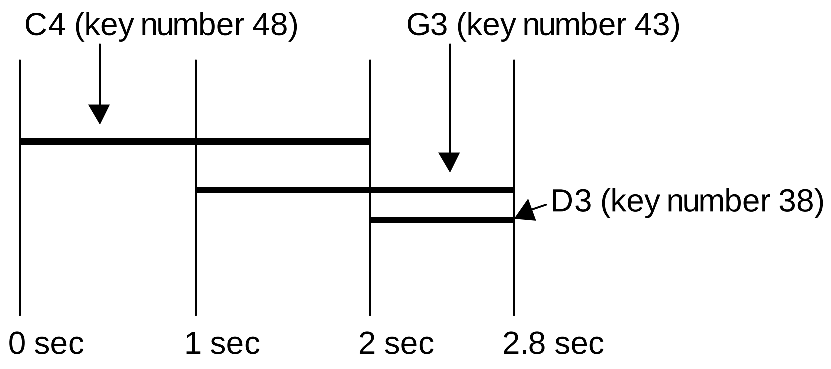


Figure 7: An Example in Piano Roll Notation

Step 5 calls *getNoteEvent*. The function *getNoteEvent* finds the next note event from the input MIDI stream. The MIDI stream is currently an input file and *getNoteEvent* reads the file until the next "note on" or "note off" is encountered. It then returns *deltaTime*, *keyNumber*, *velocity* and *eventType* for the event. If no note event is found, *getNoteEvent* returns "-1."

To understand, *MIDI2PallSm* from Steps 5 through 19, consider the generation of P_{all} and S_m from the example music in Figure 7.

The first "note on" event is read in Step 5. The function *getNoteEvent* returns *deltaTime* = 0, *keyNumber* = 48, *eventType* = "note on." HarmAn then sets *time* and p_i to 0. The *pitchClass* for the event is event is 0. Since $p_i = p_{i-1}$, no new entries are made to P_{all} or S_m. To show a "C" is currently sounding, element 0 of the *state* array is then incremented by 1, making *state* = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].

The next note event read in is also a "note on," and *getNoteEvent* returns *deltaTime* = 1, *keyNumber* = 43, *eventType* = "note on." The variables *time* and p_i are set to 1 in steps 7 and 8. Now, $p_i <> p_{i-1}$, so the code to add elements to P_{all} and S_m is run. P_{all} is set to {0, 1}. Segment $s_{<p_{i-1}, p_i>}$ is set to <0, 1, "no label", [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] > and S_m to $\{s_{<p_{i-1}, p_i>}\}$. Now that the segment ending at p_i has been generated, *state* is updated to [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] to show a "G" is sounding.

The next note event read is *deltaTime* = 1, *keyNumber* = 48, *eventType* = "note off." The variables *time* and p_i are set to 2. Since $p_i <> p_{i-1}$, P_{all} is set to {0, 1, 2}. Segment *s* is set to <1, 2, "no label", [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] > and S_m to $S_m \cup \{ s_{<p_{i-1}, p_i>}\}$. Since,

pitchClass is 0 and *eventType* = “note off”, *state* is updated to
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0].

The next note event read is *deltaTime* = 0, *keyNumber* = 38, *eventType* = “note on.” The variables *time* and p_i are set to 2. Since $p_i = p_{i-1}$, no new elements of P_{all} or S_m are generated. The *pitchClass* is 2, so *state* is updated to [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0].

The next note event read is *deltaTime* = 0.8, *keyNumber* = 43, *eventType* = “note off.” The variables *time* and p_i are set to 2.8. Since p_i <> p_{i-1}, P_{all} is set to {0, 1, 2, 2.8}. Segment *s* is set to <2, 2.8, “no label”, [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]> and S_m to $S_m \cup \{s_{<p_{i-1}, p_i>}\}$. Since, *pitchClass* is 7 and *eventType* = “note off”, *state* is updated to
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0].

The final event read is *deltaTime* = 0, *keyNumber* = 38, *eventType* = “note off.” The variables *time* and p_i are set to 2.8. Since $p_i = p_{i-1}$, the sets P_{all} or S_m are not updated. The *pitchClass* is 2 and *eventType* = “note off”, so *state* is updated to
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].

The final output *MIDI2PallSm* generates on the example in Figure 7 is as follows.

P_{all} = {0, 1, 2, 2.8}

S_m = {<0, 1, “no label”, [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] >,
<1, 2, “no label”, [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] >,
<2, 2.8, “no label”, [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]>}

The time required to generate P_{all} and S_m is linear with respect to the number of notes. This can be seen by noting that for any “note on” or “note off” event, a fixed number of steps are taken in the code. Each of these steps requires a fixed amount of time. Thus, the maximum number of steps taken in the process of generating P_{all} and S_m is related to the number of notes by a constant factor.

5.2 Finding a Good Partitioning

Once the initial partitioning, P_{all}, is created, it is assumed to be a good partitioning, unless another partitioning is proven to be better. To find a better partitioning, HarmAn goes through the partition points of P_{all} in order and determines whether the overall score of the piece would be increased by the removal of each partition point.

Let $P_{good} = P_{all}$ and $i = 2$. Let p_i be the partition point in P_{good} under consideration. HarmAn compares the current best partitioning, P_{good}, with $P_{good} - p_i$. If the scoring algorithm determines that $P_{good} - p_i$ is a better partitioning than P_{good}, then p_i is removed from P_{good}. The system then gets a new p_i, and repeats the process. Figure 8 contains pseudo code describing this step in greater detail.

As can be seen from Figure 8, HarmAn scores the segment on either side of partition point p_i. It then sums the states of the two segments by adding $state_{<pi-1,pi>}[j]$ to $\mathrm{state}_{<pi, pi+1>}[\mathrm{j}]$. Note that the values in $state_{<pi-1,pi>} + state_{<pi, pi+1>}$ may not actually represent the count of notes of each pitch class found in the segment <p_{i-1}, p_{i+1}>. Consider the case where a single note is held while a note changes in another voice. The first two minimal segments (i.e. the first half-beat) of the Beethoven example in Figure 2 show this situation. The piece begins with two “A flats” and one “C.” These are the notes playing in segment <p_1, p_2>. Then, one of the “A flats” moves to an “E flat” in the next quarter-beat (i.e. segment <p_2, p_3>). The union of these two segments, <p_1, p_3>, contains two “A flats”, one “C” and one “E flat.” HarmAn represents the states for these segments as follows.

$$state_{<p1,p2>} = [1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0]$$

$$state_{<p2,p3>} = [1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]$$

$$state_{<p1,p3>} = state_{<p1,p2>} + state_{<p2,p3>} = [2, 0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0]$$

The state that results from this addition clearly does not represent the number of notes of each pitch class occurring during the segment <p_1, p_3>. In fact, for any non-minimal segment, a note will be represented by the number of minimal segments it spans within the segment in question. Thus, the held “C” in the example is represented by a value of 2 in $state_{<p1,p3>}$. Similarly, the slot for “A flat” has the value 3 in $state_{<p1,p3>}$ because the “A flat” held across minimal segments <p_1, p_2> and <p_2, p_3> receives a point for each of the minimal segments in which it is present, and the sixteenth note “A flat” is receives a point for being in <p_2, p_3>. Note that, although the low “A flat” is actually held for a total of four minimal segments, it only has a value of two in the *state* array, since the interval under consideration only includes two minimal segments of the four in which the “A flat” sounds.

This system for counting note weight by the number of minimal segments spanned gives more importance to held sonorities and ensures that the segment labeling generates higher scores when notes spanning several minimal segments are present. This causes two adjacent segments which spell out the same chord and share a held note (such as segments <p_1, p_2> and <p_2, p_3> from the Beethoven example) to receive a score at least as high when joined as they do when separate. This is important due to the way HarmAn searches for a good partitioning.

If the score of the sum of two adjacent segments is equal to or higher than the sum of their individual scores, then HarmAn deems it has found a label that better explains the notes in the two segments as a single unit and partition point separating them, p_i, is removed from P_{good}. Similarly, the segments <p_{i-1}, p_i > and <p_i, p_{i+1}> are replaced by a single segment <p_{i-1}, p_{i+1}> in S_{good}.

```
GIVEN:    $P_{all}$,   an ordered set of all partition points.
                       Each $p_i \in P_{all}$ is a real number corresponding to the time of a "note on" or "note off" event

          $S_m$        an ordered set of minimal segments associated with $P_{all}$
                       Each $s_i \in S_m$ is of the form < $p_{i-1}$, $p_i$, label, score, state>, begins at $p_{i-1}$ and ends at $p_i$

RETURN:   $P_{good}$,  an ordered set of partition points representing a good partitioning of $M$.
                       $P_{good} \subseteq P_{all}$

          $S_{good}$   an ordered set of minimal segments associated with $P_{all}$
                       Each $s_i \in S_{good}$ is of the form < $p_{i-1}$, $p_i$, label, score, state>, begins at $p_{i-1}$ and ends at $p_i$
                       where $p_{i-1}$, $p_i$ are partition points $i$-1 and $i$ in $P_{good}$
METHOD

1.      $P_{good}$   := $P_{all}$
2.      $S_{good}$   := $S_m$
3.      $i$          := 2
4.      $s_i$        := LabelAndScoreSegment ($s_i$)
5.      WHILE $i < |P_{good}|$
6.              $s_{i+1}$    := LabelAndScoreSegment ($s_{i+1}$)
7.              $s_{new}$    := < $p_{i-1}$, $p_{i+1}$,"no label", 0, $state_{i+1} + state_i$>
8.              $s_{new}$    := LabelAndScoreSegment ($s_{new}$)
9.              IF $score_{new} \geq (score_{i-1} + score_i)$
10.                     remove $p_i$ from $P_{good}$
11.                     remove $s_{i+1}$ from $S_{good}$
12.                     $s_i$ : = $s_{new}$
13.             ELSE
14.                     $i := i + 1$
15.             END
16.     END
```

Figure 8: FindGoodPartitioning

The pseudo code in Figure 8 shows that the partition points p_2 through $p_{|Pall|-1}$ are evaluated by HarmAn in the course of generating the final good partitioning. Evaluation of each partition point, p_i, requires the scoring of three segments, $<p_{i-1}, p_i>$, $<p_i, p_{i+1}>$ and $< p_{i-1}, p_{i+1}>$. Once the initial partition point, p_2, is evaluated, only two new segments need to be scored and labeled to evaluate each partition point, since the segment $<p_{i-1}, p_i>$ can be reused from the previous step. Thus, the total number of segments scored and labeled by HarmAn is expressed by the following formula.

$$2(|P_{all}|-2)+1 = 2|P_{all}|-3 = \text{number of segments scored by HarmAn}$$

The function "LabelAndScoreSegment " (called in Steps 6 and 8 of "FindGoodPartitioning") is described in Section 5.3. There, we show that HarmAn labels a segment s in c steps, where c is a constant. This allows us to express the number of steps required to generate a good partitioning, P_{good}, as follows.

$$c(2|P_{all}|-3) = \text{the number of note evaluations.}$$

At most, each note generates two partition points, one for the start and one for the end of the note. Thus, there can be no more than 2|M| partition points.

$$2|M| >= |P_{all}|$$

Substituting 2|M| for |P_{all}| results in the following limit on the maximum number of steps needed to generate P_{good}

$$O(c(4|M| -3))$$

The maximum number of steps required to analyze any piece of music is related by a constant to the number of notes.

A lower bound for the number of steps that take place in the course of labeling M is given by the case in which all notes in M start and end together. In this case, the piece has only one segment with |M| notes in it and the number of steps required for the scoring of a single segment, c, becomes the number of steps required for the whole piece.

5.3 Segment Labeling and Scoring

HarmAn's method for labeling segments uses the templates described in Table 2 to find the best-matching label for the set of notes in a particular segment. In the current implementation, templates for "major", "minor", "augmented" and "diminished" triads are used, as well as a template for the "major-minor" (a.k.a. "dominant") 7th chord. These templates are matched against the set of notes present in a particular segment $<i, j>$ as represented by the *state* array for that segment and the template whose root pitch-class and chord quality best match the notes has its name returned as the label for the segment. The score for a particular combination of template and state is calculated by performing the steps in Figure 9.

1. If all elements in $state_{<i,\,j>}$ are 0, then the segment is a rest. Give one point and quit.
2. Sum the values of the elements of $state_{<i,\,j>}$ whose index number matches a template element
3. Subtract the values in the elements of $state_{<i,\,j>}$ whose index number does not match any element of the template.
4. Add ½ the value of the element of $state_{<i,\,j>}$ whose index number is the root class of the template.
5. Multiply the result by the number of template elements that correspond to the index of an element of $state_{<i,\,j>}$ containing a value greater than 0.

Figure 9: Calculating a Template's Score

Step 1 gives a positive point value to segments containing only rests. Assigning a positive value to segments containing only rests prevents the removal of the partition points defining the rest segment, since unifying any segment with a rest segment lowers the overall score. Rests often form a natural division point between phrases. Due to this we deemed it undesirable to unify segments across rests.

Steps 2 and 3 give the initial score for the segment through a count of the number of template matches minus the number of template misses.

Step 4 causes the system to favor labels that use one of the pitches in the segment as the root of the chord. Without Step 4, a segment containing “A” and “C” would return a label of “F major” rather than “A minor”, due to HarmAn’s preference for major chords over minor (see Figure 11).

Step 5 enables the unification of arpeggiated chords through giving higher values to templates that have several of their elements matched by the data. Take the example of two segments containing “C” and “E”, respectively. HarmAn labels them “C major” and “E major” with scores of 1.5 and 1.5. Without Step 5, the union of the two segments would be have a score of only 2.5, and unification would not take place, since the unified segment has a lower score than the sum of the two individual segments. Step 5 doubles the score of the unified segment since two elements in the template for “C major” are matched. The resulting score of 5 is higher than the sum of the separate scores, and the two segments are unified.

Consider an example application of the steps in Figure 9 on the template for an F minor triad. Let the segment be $<p_1, p_3>$ from the Beethoven example in Figure 2. This segment has the following state (derived in Section 5.2).

$$state_{<p1,p3>} = [2, 0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0]$$

The root, r, is an “F” (pitch class 5) and the template for a minor triad is mod12(r+0, r+3, r+7). The elements of the *state* array which match the template are $state_{<p1,p3>}[5]$, $state_{<p1,p3>}[8]$, and $state_{<p1,p3>}[0]$.

Step 1 does not apply, since there are elements in $state_{<p1,p3>}$ with a value > 0.

Steps 2 and 3 sum the values of template elements minus those not matching the template.

$score = \text{sum}(state_{<p1,p3>}[5,8,0]) - \text{sum}(state_{<p1,p3>}[1,2,3,4,6,7,9,10,11])$ The root class is 5, which is empty, so there is nothing to add in Step 4.

There are two template elements with positive values in the state array, so Step 5 is

$$score = score * number\ of\ template\ elements\ matched = 4 * 2 = 8$$

Thus, the score for an F minor triad on the segment $<p_1, p_3>$ is 8.

In practice, all combinations of template and root class must be scored in order to determine the best (i.e. highest scoring) label for a particular segment. The remainder of this section outlines how this is done.

GIVEN: *state* , an integer array of size 12 indexed from 0 to 11.
The value of an element, *i*, of *state* corresponds to the number of notes of pitch class *i* present in the segment times the number of minimal segments during the current segment in which each note sounds.

template, an array of integers from 0 through 11 representing distances in pitch-class space from a root class. A *template* may have from 1 to 12 elements in the array, given only 12 possible pitch classes.

RETURN: *bestRoot,* an integer from 0 to 11 corresponding to the pitch class of the highest scoring root

bestScore, a real number corresponding to the confidence in the root choice

METHOD

```
1.  bestScore := -100
2.  bestRoot := 0

3.  FOR root := 0 to 11
4.      score := 0

5.      myTemplate := mod12(root + template)       % set template relative to the root

6.      score := sum(state(myTemplate))            % sum the values in elements
                                                   % that match the template
7.      score := score + state(root)/2             % add extra 50% for roots
8.      score = score - sum(state(notInTemplate))  % subtract the sum of the
                                                   % values not in the template
9.      numberOfElementsMatched = sum(state(myTemplate) > 0)
10.     score = score * numberOfElementsMatched

11.     IF score > bestScore
12.         bestScore := score
13.         bestRoot := root
14.     END
15. END
```

Figure 10: FindBestRoot

Given a template for a chord-quality has been determined, the best root and score for that root are found using a method outlined by the pseudo code in Figure 10. This algorithm takes as input a template similar to those described in Table 2 and the *state* array of the current segment. It returns the highest scoring root pitch class and a score for that combination of template and root class.

Note that the number of steps taken in Figure 10 depends on the number of elements in the template rather than the number of notes present in the input segment. Since no template may have more than 12 elements (one for each pitch class), the number of steps taken in the course of finding the best root is bounded by a constant value, O(c).

Ties between the scores generated for two templates are resolved through the application of the preference rules described in Figure 11. Preferences are transitive. Thus, major triads are preferred to minor triads and minor triads are preferred to augmented ones, so major triads are preferred to augmented triads.

The chord quality preference rules in Figure 11 are intended to capture the relative likelihoods of the chord qualities (“minor,” “augmented,” etc.) in tonal music. Of course, the likelihood of a particular chord quality will vary with the kind of music analyzed, but the preferences we have selected form a good general rule of thumb for tie resolution. The pitch class preference rule is essentially arbitrary and is intended to provide a simple, predictable tie resolution method when there is no other way of choosing between two labels.

Prefer major triad to minor triad

Prefer minor triad to major-minor 7th

Prefer major-minor 7th to diminished triad

Prefer diminished triad to augmented triad

Prefer lower pitch-class numbers to higher pitch-class numbers

Chord-quality preferences take precedence over pitch-number preferences

Figure 11: Preference Rules for Template Tie Resolution

The function "LabelAndScoreSegment ", shown in Figure 12, outlines the steps HarmAn takes in labeling and scoring a segment. If the segment is empty, then it returns a score of one and reports that there is no chord in the segment. Otherwise, the program goes through each combination of root and template, comparing their scores and returning the highest scoring root and label. Computing best root and score for an individual template is performed by "FindBestRoot," described in Figure 10. Ties are resolved by the function "IsPreferredTemplate", which compares the best template so far with the current one, using the preferences in Figure 11. This function is not written out in pseudo code, since it is simply a set of "if – then" statements embodying the preferences in Figure 11.

Note that nothing in this approach limits templates to triadic harmonic structures. Pentatonic scales, constructions based on fourths or any other structure can be searched for simply by introducing a template for the structure in question and establishing a rule for resolving ties between the new template and existing ones.

GIVEN: A segment, s_i, of the form < p_{i-1}, p_i, *label, score, state*>, beginning at p_{i-1} and ending at p_i

RETURN: A segment, s_{out}, of the form < p_{i-1}, p_i, *label, score, state*>, beginning at p_{i-1} and ending at p_i

METHOD

```
1.   MajIntervals := [0 4 7]
2.   minIntervals := [0 3 7]
3.   dimIntervals := [0 3 6]
4.   AugIntervals := [0 4 8]
5.   Dom7Intervals := [0 4 7 10]
6.   template := [dimIntervals , AugIntervals , Dom7Intervals, minIntervals , MajIntervals ];

7.   topIndex := 0,
8.   topScore := 0,
9.   topRoot := 0

10.  IF all elements of $state_{in}$ = 0
11.      $s_{out}$ := < $p_{i-1}$, $p_i$,'rest', 1, $state_{in}$>
12.  ELSE
13.      FOR templateIndex := 1 to 5
14.          newBestTemplate := FALSE
15.          myTemplate := template(templateIndex)
16.          [root, score] := FindBestRoot ($state_{in}$, myTemplate)

17.          IF score > topScore
18.              newBestTemplate := TRUE
19.          ELSEIF ( score =  topScore )  AND IsPreferredTemplate(templateIndex, topIndex)
20.              newBestTemplate := TRUE
21.          END

22.          IF newBestTemplate = TRUE
23.              topScore := score
24.              topIndex := index
25.              topRoot  := root
26.          END
27.      END
28.      score := topScore
29.      label := MakeLabel(topRoot, topIndex)
30.      $s_{out}$ := < $p_{i-1}$, $p_i$,label, score, $state_{in}$>
31.  END
```

Figure 12: LabelAndScoreSegment

The approach for labeling a segment outlined in Figure 12 takes a fixed amount of time for each step save step 16, which calls FindBestRot. It was shown earlier that the maximum number of steps taken by FindBestRoot is limited to a fixed amount as well. Thus, the maximum number of steps required to label a segment is also fixed.

LabelAndScoreSegment returns a result in constant time

6 An Example: Beethoven Sonata Pathetique, 2nd Movement, Measure 1

To better understand the HarmAn approach to finding a good partitioning, consider an analysis of the first measure of the second movement of Beethoven's Sonata Pathetique. Figure 13 shows the first measure of the Beethoven example with its partition points labeled.

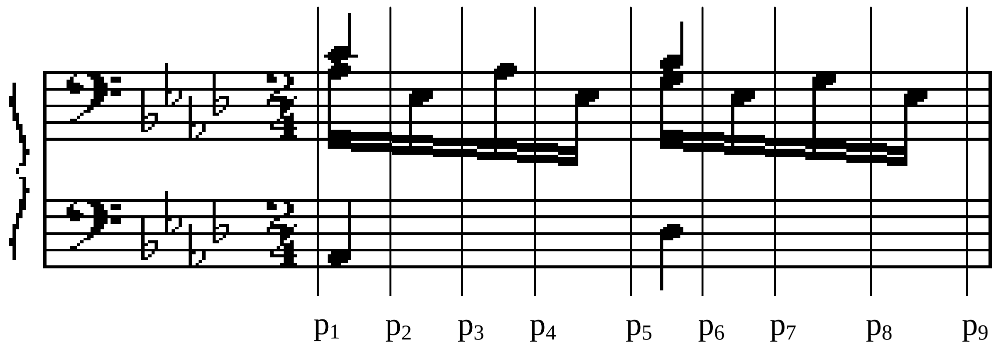


Figure 13: Beethoven, Sonata Pathetique, Op 13, 2nd Mvmt., measure 1

Recall that the partition points are sorted in temporal order, so a partitioning is represented by a binary number where bit i indicates whether partition point i should be used to partition two segments. Recall, also, that only the bits for the second through the penultimate partition points are needed to uniquely identify a partitioning for a piece of music. There are nine partition points in the first measure of the Beethoven example, making $|P_{all}| = 9$. Thus, the initial partitioning generated by HarmAn is represented by "1111111."

Table 3 : Partitionings Considered in Analysis of Example

	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7
Current	1111111	0111111	0011111	0001111	0001111	0001011	0001001
New	0111111	0011111	0001111	0000111	0001011	0001001	0001000

Table 3 shows the series of partitionings considered by HarmAn in the analysis of the Beethoven example. There are seven rounds of comparison. For each round, the best partitioning found so far is in the *Current* row. The new partitioning under consideration is in the *New* row. The bit representing the partition point under consideration in the current round is shown in a larger font than the rest of the bits. The higher-scoring partitioning in each round is highlighted with a gray background. To see how the binary numbers in Table 3 map onto partitionings, consult Figure 4, which shows two partitionings from Table 3 in standard notation.

Table 4 shows the label and score for every possible segment in the first measure of the Beethoven example. The vertical key gives the starting partition point for each segment. The horizontal key gives the ending partition point for each segment.

Table 4 : Segment Labels and Scores

	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	A♭ Maj 8	A♭ Maj 22.5	A♭ Maj 34.5	A♭ Maj 45	A♭ Maj 36	A♭ Maj 33	A♭ Maj 24	E♭ Dom 24
p_2	-	A♭ Maj 10.5	A♭ Maj 22.5	A♭ Maj 33	A♭ Maj 24	A♭ Maj 21	E♭ Dom 22	E♭ Dom 36
p_3	-	-	A♭ Maj 8	A♭ Maj 22.5	A♭ Maj 13.5	E♭ Dom 12	E♭ Dom 24	E♭ Dom 38
p_4	-	-	-	A♭ Maj 10.5	E♭ Dom 10	E♭ Dom 24	E♭ Dom 36	E♭ Dom 50
p_5	-	-	-	-	G dim 10.5	E♭ Dom 26	E♭ Dom 38	E♭ Dom 52
p_6	-	-	-	-	-	E♭ Dom 10.5	E♭ Dom 26	E♭ Dom 40
p_7	-	-	-	-	-	-	G dim 10.5	E♭ Dom 26
p_8	-	-	-	-	-	-	-	E♭ Dom10.5

As stated previously, HarmAn considers partition points in the order in which they occur in the music. In the first round of comparison, the partitioning 1111111 (P_{all}) is compared to partitioning 0111111 ($P_{all} – p_2$). This is done by comparing the scores of segments <p_1, p_2>, < p_2 , p_3> and <p_1 , p_3>. Here, the score of <p_1 , p_3> is 22.5 and the sum of the scores for <p_1, p_2> and < p_2 , p_3> is 18.5. Since the unified segment has a higher score, partition point p_2 is discarded (i.e. its bit is set to 0) from the partitioning for this reason.

In Round 2, HarmAn compares partitioning 0111111 with partitioning 0011111. This is done by comparing segments <p_1 , p_3>, < p_3 , p_4> and <p_1 , p_4>. As in the previous round, the unified segment scores higher than the sum of the scores of the two component segments and p_3 is discarded from the partitioning.

In Round 3, HarmAn compares <p_1 , p_4> + <p_4 , p_5> vs. <p_1 , p_5>. In this case, the values are equal, but the system chooses to remove the partition point due to a preference for longer segments over shorter ones. This is done whenever the sum of the two separate scores equals the unified score.

Round 4 finds HarmAn comparing partitionings 0001111 and 000011 by looking at segments <p_1, p_5>, < p_5 , p_6> and <p_1 , p_6>. The sum of the scores of <p_1, p_5> and

<p_5, p_6> is 55.5, while the score of the segment <p_1 , p_6> is only 36. Removing the partition point p_6 would reduce the score, so the partition point remains (is assigned a 1) and the system moves on to the next round.

In Round 5, HarmAn chooses between partitionings 0001111 and 0001011 by comparing <p_5, p_6> + < p_6, p_7> with <p_5, p_7>. On basis of this, p_6 is deemed to not to be harmonically relevant and is removed from the partitioning.

In the final two rounds, HarmAn considers the partition points p_7 and p_8. In both cases, the score for the unified segment is higher than the sum of the two seperate segments' scores. Due to this, both p_7 and p_8 are removed from the final partitioning.

Since the initial and final partition points are always in every partitioning, the system is done once it has considered points p_2 through p_8, leaving partitioning 0001000 (illustrated in Figure 4) as the final winner, with the first segment labeled as "A flat major" and the second segment labeled "E flat dominant."

7 Evaluation of HarmAn

7.1 Time Complexity of the HarmAn approach

Section 5.1 shows that HarmAn generates initial partitioning P_{all}, and segmentation S_m in a number of steps proportional to the number of notes in a piece of music, M.

$$\text{Steps to generate } P_{all} \text{ and } S_m = c|M|\text{, for some constant } c$$

Section 5.2 shows that the maximum number of steps required to generate the final partitioning, P_{good}, from P_{all} is given by the following expression, where d is a constant.

$$d(4|M| - 3) \geq \text{maximum number of steps to generate } P_{good}$$

Adding the time required for each of these steps results in the following limit on the number of steps HarmAn requires to create a final partitioning with labeled segments.

$$k|M| \geq c|M| + d(4|M| - 3) \geq \text{steps to parse a piece of music, where } k = c + 4d$$

7.2 The Number of Partitionings Searched by HarmAn

HarmAn generates a final partitioning in time $k|M|$. The full space of possible partitionings is $2^{|Pall|-1}$, which is at most $2^{2|M| - 2}$. This provides an indication that HarmAn does not search the entire space of partitionings in the course of generating a result. In fact, it does not search the entire space.

$$2|M| - 1 \geq |P_{all}| - 1 = \text{the number of partitionings searched by HarmAn}$$

Each partition point is considered in order of its occurrence in the music. Once a partition point has been deemed relevant or not, it is not reconsidered later. The number of partitionings considered by the system is equal to $|P_{all}|$-1, since two partitionings are considered for partition point p_2 and one new partitioning is considered for each subsequent partition point up through $p_{|Pall|-1}$.

This number is quite a bit smaller than the full space of $2^{|Pall|-1}$ possible partitionings. This great reduction in the number of partitionings considered is a key element in HarmAn's ability to analyze a piece of music in tractable manner. One may ask whether it is reasonable to prune the space of partitionings and segments in the way HarmAn does. We think that it is reasonable to do so.

Assume that a higher-scoring template corresponds to one closer to a human expert's choice of label for a segment of the music. HarmAn's processing of the partition points in P_{all} results in a monotonically non-decreasing overall score. This can be seen from the algorithm for finding a good partitioning in Figure 8, where a partition point p_i is only removed from P_{good} if $score(< p_{i-1}, p_{i+1}>) \geq score(< p_{i-1}, p_i>) + score(<p_i, p_{i+1}>)$. Thus, a partition point is removed only if the resulting score is not decreased. This guarantees that the final segmentation S_{good}, will have at least as high a total score as the initial segmentation of minimal segments, S_m. This does not assure that S_{good} has the highest possible score, only that it is at least as good as when the process begins.

From this, we can say that HarmAn's final partitioning, P_{good}, and its labeled segmentation, S_{good}, are no further from a human expert's partitioning and labeling of a piece than are P_{all} and S_m.

Of course, this does nothing to guarantee that the parse resulting from the start-to-finish approach used by HarmAn generates an answer anywhere near the highest scoring one. The argument in favor of this approach is that music is an art form that unfolds in time from start to finish. A composer who wishes to write pieces that are decipherable by the listener must create structures that can be understood in a start-to-finish way with limited backtracking to reconsider previously heard passages. Also, the expectation for what comes next is determined by what has just been heard and it seems reasonable to assume that the set of likely partitionings under consideration by a human is greatly constrained by what has already transpired in the music. A good composer knows this at some level and writes music accordingly. From this, it seems likely that much music is written so that one can partition it in a start-to-finish way with no backtracking.

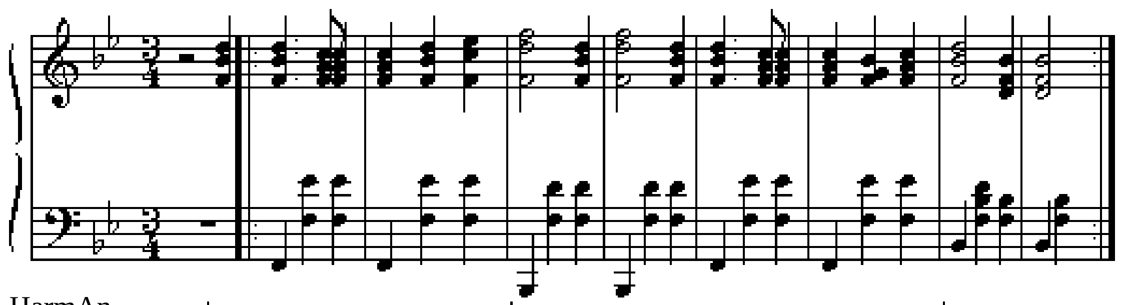
7.3 Evaluation of HarmAn on Various Pieces

If it is true that much tonal music was written to be analyzed in a start-to-finish way, and that the template scoring algorithm in Figure 9 is a good one, then HarmAn should perform well on a variety of pieces. In order to compare our results to existing work on automated analysis of harmony, we analyzed a set of pieces by Bach, Beethoven, and Schubert used in other papers. We also analyzed a number of other pieces of various textures from various periods. What follows are several examples taken from previously published papers showing our system's analysis along side the published results and one more difficult piece by Debussy, showing HarmAn's analysis of a tonally ambiguous passage.

7.3.1 Winograd : Schubert. Deutsche Tanze, Op. 33, No. 7

Winograd (Winograd 1968) approached the harmonic analysis of music through the use of generative grammars. Winograd’s work was successful in correctly labeling the harmonies, roman-numeral style, in pieces of music composed of block chords but did not address the issue of how to partition a piece into segments (i.e., likely chords) upon which to perform the harmonic analysis. This partitioning was performed by a human operator and the results of the human’s parsing were passed to the program as input. This contrasts with HarmAn’s automatic partitioning of the music.

In order to test HarmAn on music with block chord figuration and to compare our results to Winograd’s, the same pieces analyzed in Winograd’s paper were analyzed by HarmAn. An example of HarmAn’s analysis of the first eight measures of one such piece is shown in Figure 14. The results generated by HarmAn for this passage generally agree with Winograd’s analysis, with the only major disagreement being in the placement of the “F7” (F dominant) chord in the second half of the sixth measure. Winograd’s system placed this chord’s beginning at the start of the seventh measure. Interestingly, a nearly identical musical situation presents itself in the second measure and here Winograd’s system agrees with our analysis.



HarmAn B♭Maj F7 B♭Maj F7 B♭Maj

Winograd, Jazz notation B♭ Maj F7 F Maj B♭ Maj F Maj F7 B♭ Maj

Winograd, orig. notation I^6_4 V7 V I I^6_4 I I^6_4 V V7 I I^6_4 I

Figure 14 Schubert. Deutsche Tanze, Op. 33, No. 7

7.3.2 Temperley and Sleator : Beethoven Sonata Pathetique, 2nd Movement

Temperley and Sleator’s (Temperley and Sleator 1999) system takes a piece of music as input and first does beat finding in a manner strongly reminiscent of Lerdahl and Jackendoff (Lerdahl and Jackendoff 1983). These beats are then used to help determine the partitioning of the piece into time spans, which are labeled as likely chords. The exact method used to determine the initial partitioning is not given in the paper except to say that segments should begin on subdivisions of the beat and should be “short” (in the range of 100-300 ms). A root is then chosen for each segment. Roots are chosen so as to prefer giving the same root name to successive chords if possible, and to prefer root-names related by a fifth, otherwise. Their system is limited to labeling the root of each segment rather than providing the full chordal spelling. The full history of preceding root names is used, along with the intervals present in the current segment, to determine the choice of the root name of each segment. The system is described as a set of preference rules, and is strongly tied to an explicit model of functional tonal harmony in its approach to analysis.

Figure 15 shows the results of analysis of the first eight measures of the second movement of Beethoven's Sonata Pathetique (Temperley and Sleator only reported the analysis of the first five measures in their paper). As can be seen from the figure, HarmAn successfully captured the correct chord roots and qualities in this passage, with the possible exception of the first half of the fifth measure. HarmAn's choice of chord in the first half of the fifth measure, while a reasonable harmonization given the context, might not be as good as a label of "G diminished" or possibly "E flat dominant". The harmonic rhythm was also captured accurately without need to infer meter or beat.

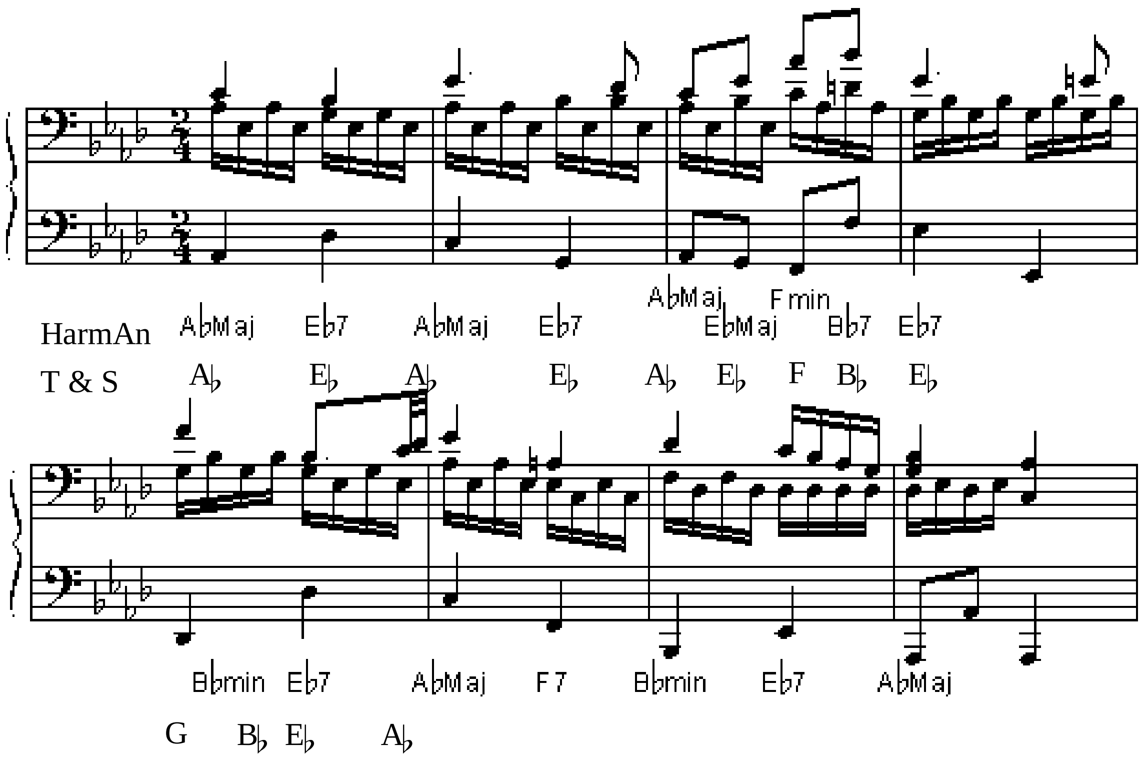

HarmAn AbMaj Eb7 AbMaj Eb7 AbMaj EbMaj Fmin Bb7 Eb7
T & S Ab Eb Ab Eb Ab Eb F Bb Eb
Bbmin Eb7 AbMaj F7 Bbmin Eb7 AbMaj
G Bb Eb Ab

Figure 15: Beethoven. Sonata Pathetique. 2nd Movement.

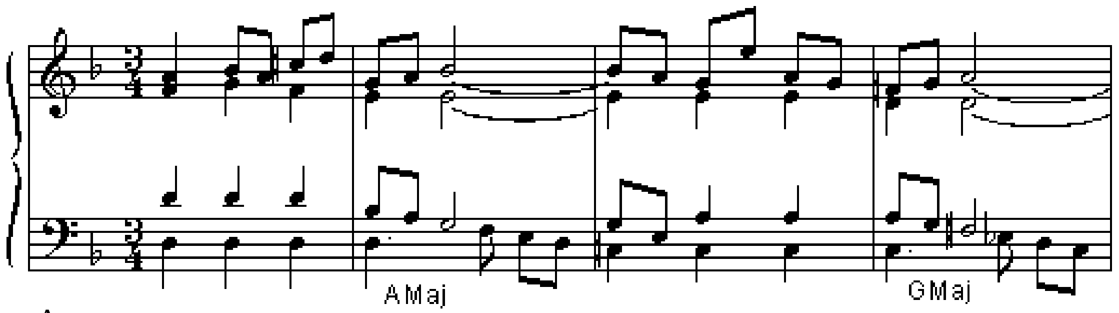
The Temperley and Sleator system did about as well as HarmAn on root finding in this example, although HarmAn also correctly identifies chord quality. Both systems agree on root spellings through the first four measures, diverging on the fifth measure. Their system finds four roots in this measure, namely "G", "B flat", "E flat" and "A flat."

HarmAn reports only two. Interestingly, HarmAn gets closer to capturing the actual harmonic rhythm of this measure even though it does not explicitly represent rhythm.

7.3.3 Maxwell : J. S. Bach, 1st French Suite, D minor Sarabande

Maxwell (Maxwell 1992) built a system that performed harmonic analysis using hundreds of preference rules for analysis of individual notes and intervals between notes. An example rule is: “RULE 22: If a vertical is unaccented AND it is tertian AND the previous vertical is tertian AND they both have the same root, THEN the vertical is subordinate to the previous vertical.”

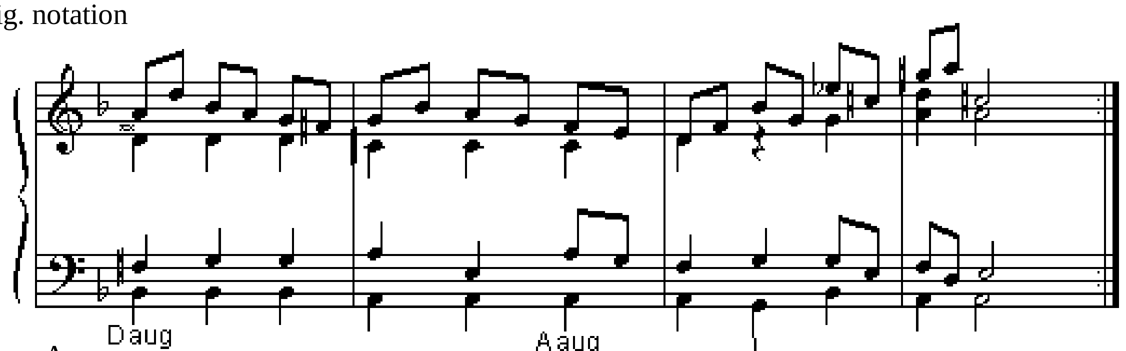
A strength of Maxwell’s work is that it addressed determining which sets of notes should be labeled as chords. Unfortunately, the paper describing the work lists only a small subset of the preference rules, and is unclear on how the rules are weighted or what control structure mediates between these rules. Maxwell did not explicitly address the issue of the computational complexity of the partitioning problem, nor what portion of the space was actually searched.



HarmAn Dmin AMaj Gmin Gmin A7 GMaj D7 D7

Maxwell, Jazz notation D min E dim 7 E dimA7 D7 D

Maxwell, orig. notation d:i ii^{o}4-2 ii^{o} V6 V4-2/iv V/iv



HarmAn Daug Gmin A7 Aaug A7 Dmin E♭7 Dmin A7

Maxwell, Jazz notation G min A7 D min G min A C# dim7

Maxwell, orig. notation iv6 V7 i6-4 iv vii^{o}4-2 V

Figure 16: J. S. Bach, 1st French Suite, D minor Sarabande

Maxwell's system analyzed a number of pieces, generating results in roman numeral-style notation. One such piece is the D minor Sarabande from J. S. Bach's First French Suite. Both HarmAn's and Maxwell's analysis of the first eight measures of the Sarabande is shown in Figure 16. In general, our system was successful, correctly capturing the correct chord-quality and root in each measure.

HarmAn differed from the Maxwell system in the second measure because a diminished 7th chord is not one of HarmAn's templates. Thus, it chose a label of "G minor." The pitch classes in "G minor" form a subset of those in "E diminished 7" so the labeling difference between the two systems on this measure is small.

The penultimate measure shows a more significant difference. Here, HarmAn labeled the last two beats of the measure as a single "E flat dominant." Maxwell's system broke the measure into three chords and labeled them as a progression that is more typical of tonal music of the period. The difference may be accounted for in the spelling of the pitches. HarmAn takes input in the MIDI format, which gives the same value to both "D flat" and "C sharp." Written music, however, often varies the spelling of the note to give an indication of its function. The written music has a "C sharp" in the third beat of the measure. Such a spelling precludes using this note as part of an "E flat dominant" chord and forces a different interpretation.

HarmAn's analysis generated a couple of spurious "glitch" chords lasting an eighth-note each. The "A major" on the second half of the first beat of measure two is an example of just such a chord. This chord is actually two passing tones and neither Maxwell's system nor most human analysts would chose to identify either the "A major" in the second measure or the "G major" in the fourth measure as structurally significant harmonies. The generation of such individually labeled short segments is a direct result of the local nature of the HarmAn approach.

The issue of context is a tricky one, however. Consider the first beat of measures five and eight. In both measures, Maxwell's system ignores what we consider to be a significant chord on the first beat of the measure, which HarmAn detects and correctly labels. In general, though, both systems generated correct analyses of the passage. This is notable given the fact that the number of rules used by HarmAn in determining the harmonic structure of a piece is far smaller than the number of rules used in Maxwell's system.

7.3.4 A Tonally Ambiguous Example : Debussy, The Little Shepherd

To this point, all the music analyzed in this paper has clearly outlined triadic tonal harmonies with few non-chord tones. However, much late Romantic and early 20th century music is much more ambiguous in its statement of tonal harmonies. The music of Debussy is a good example of this. Many Debussy pieces are loosely tonal. Cadences still exist but are obscured or led up to in ways based on non-tonal structures, such as pentatonic and whole tone scales. Given the more ambiguous nature of Debussy's harmonies, his music is good for testing the limits of a harmonic analysis program.

Figure 17 shows HarmAn's analysis of the first eleven measures of Debussy's "The Little Shepherd." In this figure, there is an arrow from each segment label to the note in which the segment begins. We selected "The Little Shepherd" because it contained several interesting properties. First, the piece begins with an initial unaccompanied melody of ambiguous tonality. We were interesting in seeing the analysis HarmAn would generate of this passage. This tonal ambiguity continues when other voices are introduced in the fifth measure. The additional voices provide neither a clear tonal center nor do they clearly outline triadic harmonies. This ambiguity clears up in the second half of the sixth measure, where the music resolves to a "B major" triad.

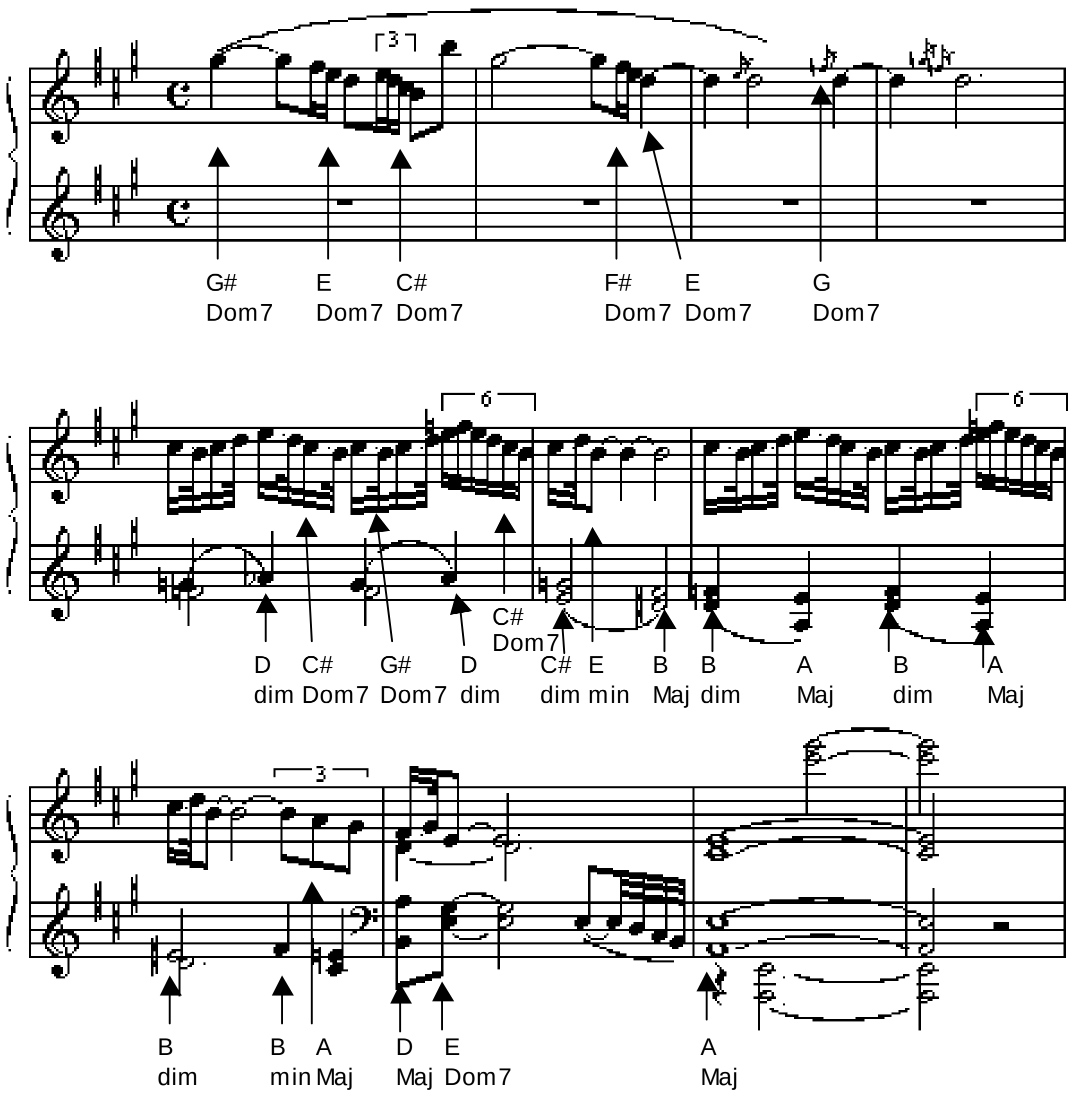


Figure 17: Claude Debussy, The Little Shepherd, mm 1-11

Measures seven and eight show another interesting feature of this piece. The melody of measures five and six is repeated with new, tonal, harmonies outlining what may be alternating "D minor" (or possibly "B diminished") and "A major" chords in the accompaniment. This allows a direct comparison of HarmAn's analysis of a non-tonal accompaniment with a more tonal accompaniment of the same melody.

The final three measures outline a "B minor 7" chord moving to an "E dominant 7" and resolving on an "A major" triad, providing a clear spelling out of these harmonies with relatively few non-chord tones.

HarmAn's analysis of the single note line in the first four measures of the piece is interesting in that, due to the templates used, it imposes a tonal harmonization upon a possibly non-tonal melodic line. The harmonization intersperses root movement by fifth (a typical tonal device) with a repeated E chord. Thus, the system infers an obscured tonal root movement pattern, a typical feature of late tonality. Unfortunately, HarmAn's selection of when to change chords is questionable, as it sometimes happens during a grace note or in the middle of a 16th note triplet.

The analysis of measure five is a bit more confused and the system repeatedly changes chord at places that a human is unlikely to. This, however, is understandable given the confusing nature of the tonal structures in the measure, and since HarmAn makes no use of metrical information in determining locations for chord changes.

The system does better on the sixth measure, coming up with a reasonable parsing of the chords as "C sharp diminished" moving to "E minor" and then to "B major."

The seventh measure of the piece shows a repeat of the melodic material in measure five with clearer harmonization in the accompaniment. HarmAn performs better on this measure than it did on measure five, correctly placing chord changes on each beat. The chords it chooses are "B diminished" and "A major" in alternation. A human might have chosen "D minor" instead of "B diminished," due to the movement of the lowest voice, but there is evidence to support "B diminished" as well, so we deemed this a good parse.

HarmAn interprets measure eight as moving from a "B diminished" to a "B major" triad and then to an "A major." Given this interpretation of measure seven, then the choice of a "B diminished" in measure seven seems good in that it makes the "B diminished" triad part of a repeated pattern extending across both measures.

HarmAn misreads the first chord of measure nine as a "D major" triad instead of a "B minor 7". This is because no template was provided for the "minor 7" chord, making the system incapable of producing it as a response. Since all notes in "D major" are in the "B minor 7" chord, we deem it an acceptable error.

The final cadence of the passage is correctly captured by HarmAn as an "E dominant 7" moving to "A major." The quick run in the last beat of measure nine is ignored in favor of the held notes and the harmonic rhythm is parsed correctly.

All in all, given the intentionally obscure nature of the harmonies in the piece and HarmAn's limitation hypothesizing from only five tonal chord types, the system did quite well.

8 Conclusions and Directions for future work

The work described in this paper provides a clear formal framework upon which to build a system for harmonic analysis of music. HarmAn is based upon this theoretical framework and provides a good first approximation of the harmonic structures in a typical piece of tonal music. HarmAn does this in a clearly explained and analyzable manner, using a relative small set of rules, which do not require an understanding of the tonal context nor any metrical information to perform the analysis. The results achieved by this system compare well to those achieved by much more complex systems reported in the literature.

There are three areas for improvement of HarmAn: increasing the number of harmonic structures which HarmAn recognizes; refining the scoring method used to determine the best label for a segment; and dealing with rests.

HarmAn has only five templates for tonal structures, namely major triads, minor triads, diminished triads, augmented triads and the major-minor 7^{th} chord. Many other obvious structures, such as minor-minor 7^{th} chords, ninth chords, and various scales can easily be added to the set of templates in use This issue is a bit more complex than it appears at first, because the addition of a new template requires a rethinking of both the preference rules for choosing between templates and the scoring method for a single template.

The scoring method (see Figure 9) for a single combination of template and root was designed for current set of five tonal templates and may not provide the appropriate weightings for all kinds of template. For example, it may not be appropriate to add extra weight to a "root" class for some kinds of non-tonal structures. Tied in with this issue is the simplistic set of preference rules for selecting between templates. Currently, the result is winner-take-all, but it may be more appropriate to report the top two or three choices in ambiguous situations where, for example, it is unclear whether a segment expresses an "A minor" or a "C major" chord. We are currently investigating a variety of template scoring methods for various contexts and levels of ambiguity.

At present, HarmAn treats all rests as "iceburgs," in that rest segments are never unified with adjacent ones. This is not an ideal approach since very short rests may not signal natural divisions within the music. It would be better to consider the possibility of unifying segments across rests in a way that reflects the length (and thus, relative importance) of the rest. This can be done by producing an initial partitioning of the music which does not attempt to join segments across rests and then going back through the music and considering the removal of partition points around rest segments in order of the length of the rest. In this way, the system will tend to remove short rest segments before long ones, giving greater weight to longer rests. We are currently enhancing HarmAn to do so.

9 References

Forte, A. (1973). The Structure of Atonal Music, Yale University Press.

Lerdahl, F. and R. Jackendoff (1983). A Generative Theory of Tonal Music. Cambridge, Mass, MIT Press.

Maxwell, J. H. (1992). An Expert System for Harmonizing Analysis of Tonal Music. Understanding Music with AI: Perspectives on Music Cognition.

Smaill, A., G. Wiggins, et al. (1993). "Hierarchical Music Representation for Composition and Analysis." Computers and the Humanties **27**: 7-17.

Smoliar, S. (1980). "A Computer Aid for Schenkerian Analysis." Computer Music Journal **4**(2).

Temperley, D. and D. Sleator (1999). "Modeling Meter and Harmony: A Preference-Rule Approach." Computer Music Journal **23**(1): 10-27.

Ulrich, J. W. (1977). The Analysis and Synthesis of Jazz by Computer. Proceedings from the 5th IJCAI.

Wakefield, G. H. (1999). Chromagram Visualization of the Singing Voice. International Workshop on Models and Analysis of Vocal Emissions for Biomedical Applications, Firenze, Italia.

Wakefield, G. H. and B. Pardo (1999). Signal Classification using Time-Pitch-Chroma Representations. The Intl. Symp. on Opt. Sci., Eng., and Instr., SPIE'99, Denver, Colorado, USA.

Widmer, G. (1992). "Perception Modeling and Intelligent Musical Learning." Computer Music Journal **16**(2).

Winograd, T. (1968). "Linguistics and the Computer Analysis of Tonal Harmony." The Journal of Music Theory **12**: 2-49.