THE CHORDAL ANALYSIS OF TONAL MUSIC

BRYAN PARDO
WILLIAM P. BIRMINGHAM
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE DEPARTMENT
THE UNIVERSITY OF MICHIGAN

MARCH 28, 2001

TECHNICAL REPORT
CSE-TR-439-01

KEYWORDS
Heuristic Search, Music Analysis, Segmentation

ABSTRACT
This paper provides a theoretical framework upon which to build a system for chordal analysis of music. We establish that, in the worst case, the partitioning and labeling of harmonies in tonal music is $O(2^N)$, where N is number of notes in a piece of music. We show that, when segments of the music can be analyzed locally, and a segment scoring metric can be found that is both additive and transitive, the problem becomes $O(N^2)$. Under these constraints, the problem is cast as finding the best path through a single-source directed acyclic graph, which can be solved in O(E + V) time. This translates to a worst-case time complexity of $O(N^2)$ for our problem. We then show that the results of the $O(N^2)$ search can be closely approximated through the use of a heuristic that allows $O(N)$ time search. The results of the heuristic search are then compared to exhaustive graph search and the results of analyses by existing systems by Winograd (Winograd 1968), Maxwell (Maxwell 1992), and Temperley and Sleator (Temperley and Sleator 1999).

1 INTRODUCTION

The study of music by AI researchers has received a lot of attention in recent years, producing systems for automatic composition (Cope 1991; Mozer 1991; Todd 1991; Ebcioglu 1992; Marsella and Schmidt 1992; Smaill, Wiggins et al. 1993; Zimmermann 1995; Polito, Daida et al. 1997), performance (Johnson 1991; Todd 1992; Katayose and Inokuchi 1993; Horowitz 1995; Windsor and Clarke 1997; Dannenberg 1998), and analysis (Winograd 1968; Moorer 1975; Ulrich 1977; Smoliar 1980; Scarborough, Miller et al. 1991; Linster 1992; Maxwell 1992; Dannenberg 1993; Hoffman and Birmingham 1999). One reason why music is so interesting to study is that listening to music and performing it, which may involve composition and analysis, is a perceptual task that is distinct from our other aural communication mechanism, speech (Kraut 1992).

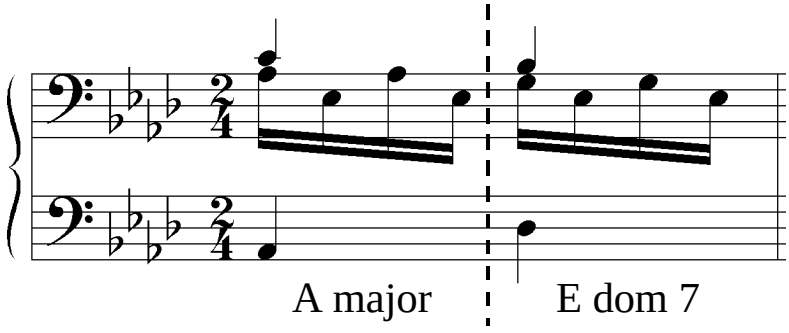

Figure 1:Beethoven, Sonata Pathetique, Op 13, Second Movement, measure 1, segmented and labeled with chord names.

In this paper, we discuss the chordal analysis of tonal music. This analysis requires that a musical stream be broken into segments that correspond to significant harmonic change.

Specifically, we search for segments that correspond to chords, labeling chords with the proper name.

We approach this task from the perspective of the listener, meaning we do not have access to the written score used by the performer in creating the music. Thus, the key signature, metrical information, and grouping information (e.g., bar lines and beams) given by the score are not available to us.Further, a performer may introduce notes that are not in the score, or may remove notes that are in the score (Heijink, Desain et al. 2000). These things, along with performer-induced timing variation, informally describe the chordal-analysis task.

In chordal analysis, *segmentation* is roughly analogous to the computer-vision task of finding lines in an image. We have developed an algorithm for segmentation that makes minimal commitments to (musical) context, thus gaining flexibility over the types of music it can process.

Labeling involves giving each segment the proper name, quality, root, and inversion. In music containing only block chords, the segmentation task is trivial, and labeling is relatively straightforward. An example of this kind of music is the typical Bach Chorale. Music, however, is rarely this straightforward (see Figure 1): chords are often arpeggiated (spread out over time), incompletely stated, and interspersed with myriad *non-harmonic tones* (NHTs) in forms such as trills and appoggiaturas. All of these things confound the task.

Yet, it is essential that we create chordal-analysis algorithms that are fast and generate high-quality results. The vast majority of Western music uses tonal-chord structures based on triadic harmonies and their elaborations as a basic structural feature. Central to the understanding of any tonal piece of music is an understanding of what chords (harmonies) are used in the piece. Thus, chordal analysis lies at the heart of all computer-based musical activities, including computer-based composition and computer-based performance.

Previous work in the area of automated chordal analysis of music (Winograd 1968; Smoliar 1980; Maxwell 1992; Widmer 1992; Smaill, Wiggins et al. 1993; Fujishima 1999), has either avoided the segmentation task by taking segmented input or has been unclear in how segmentation has been done. Other researchers reporting on tasks that are related to chordal analysis, such as harmonizing a piece of music, describe solutions using complex rule sets (Ebcioglu 1992), without discussion of more general algorithms or problem characteristics.

In this paper, we show algorithms for chordal analysis: one for segmentation and one for chord labeling. In addition, we give a heuristic that reduces the segmentation task from $O(n^2)$ (n is number of points of possible harmonic change) to O(n), while giving up very little solution quality. These algorithms use only the necessary pitch relationships for labeling, and make no use of other *contextual* information, such as composer style and meter variation, and work across the full range of tonal music. In addition, these algorithms are compact and fast. While the algorithms may make mistakes, they are

reasonable ones, and can be resolved with some musical knowledge. Our intention is that these algorithms be "general" algorithms from which more specialized algorithms can be constructed. Moreover, we formally describe the chordal-analysis task.

2 NOTATION AND TERMINOLOGY

In this section, we introduce notation needed for the description of our algorithms.

2.1 THE NOTE

We are interested in analyzing a performance of a piece of music. We work only with notes, which are pitches on an equal-tempered, chromatic scale that have a well specified start (onset) and end (off) time. Since we are primarily concerned with tonal structures, this assumption is innocuous.

In the audio domain, a pitch class represents a set of harmonic sounds whose fundamental frequencies are related by a power of two. 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 has doubled, the *octave* increases by one. Thus, A 440 is two octaves above A 110.

Table 1:Pitch Class Names and Numbers

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

Equal temperament is the most common tuning in modern Western music and the one that we assume. Equal temperament divides an octave into 12 pitch classes, 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 mapping of pitch-class names to an integer representation of pitch class is given in Table 1.

Let a *note, n,* be 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 zero through 11 representing the pitch class of note *n.*

- *octave* is an integer from zero through 11 representing *n's* octave.

The first "C" in Figure 1 is a note and is represented by 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 length of time where no pitch sounds (see Section 2.6).

2.2 THE MUSIC PERFORMANCE

A piece of music, *M,* is a set of notes ordered by *start* time. M_x denotes a particular performance of a piece of music, where *x* is the label for the performance.

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

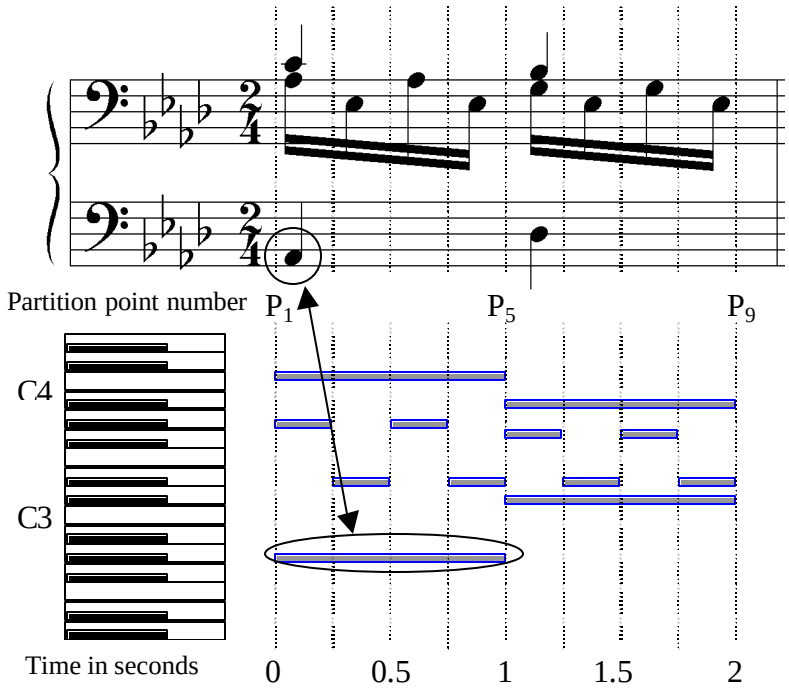

Figure 2: $M_{beethoven1}$ in Standard and Piano Roll notation

Consider the Beethoven excerpt in Figure 1. Let the performance $M_{beethoven1}$ be a realization of this excerpt, assuming a tempo of one quarter note per second (we will use this as the default tempo for all examples relating to the Sonata Pathetique). We then have the following:

$M_{beethoven1}$ = {<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>}

Figure 2 represents $M_{beethoven1}$ in standard and piano-roll notation, where each note is represented by a line. Vertical position represents the pitch of the note, length represents the duration and horizontal position represents time since the beginning of the piece of music. The note <0, 1, 9, 2> is circled in both piano-roll and standard notation to show the correspondence between them.

2.3 TIME AND METER

In this paper, *beat* usually refers to the written meter of a piece of music (e.g., ¾ time). Note, however, that there is no explicit reference to beat or metrical information in our definition of a note: the time element of a note is expressed through start and end times. Nor is there any explicit reference to metrical information in any structure based on notes. This allows music without a single metrical pulse to be easily represented and manipulated. Moreover, we do not represent nor use metrical information in our analysis.

All timing information is defined by the number of seconds since start of the earliest sounding note of the piece. The minimum value for time is zero, and the maximum value is the end time 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 (realization) of that piece. Different performances may result in timing variations that change the definition of the piece.

The choice of continuous rather than discrete time was made to avoid basing our representation on an arbitrary underlying quantization. This both simplifies things and eliminates unnecessary artifacts that may be introduced during quantization. The closest we come to a quantum is the *minimal segment* (defined in Section 2.6), which has a duration that varies with the tempo and density of notes.

2.4 STATE OF THE PIECE

Recall that a note is a four tuple of the form <*start, end, pitch class, octave*>. Let a *pitch* be a duple of the form <*pitch class, octave*>. Thus, the pitch of a note, *n*, is the final two elements of the four tuple defining a note. We refer to the pitch of a note, *n*, as $pitch_n$.

The state of the music at time *t*, $State_t$, is the set of pitches derived from the notes of M sounding at time t: $State_t = \forall pitch_n \mid (n \in M) \wedge (start_n < t) \wedge (end_n > t)$

Taking $M_{beethoven1}$, from Figure 2, the set of notes sounding at time 0.3 is

$$\{<0, 1, 0, 4>, <0, 1, 9, 2>, <0.25, 0.5, 3, 3>\}$$

Thus, the state for time 0.3 is the following…

$$State_{0.3} = \{< 0, 4>, < 9, 2>, < 3, 3>\}$$

Our analysis system, HarmAn, uses only the pitch class of the notes sounding at time t. For this reason, HarmAn, represents state as a 12-element tuple indexed by pitch-class number (0 through 11). Each element, *i*, gives the count of notes of pitch class *i* sounding at time *t*. In this representation, $State_{0.3}$ is represented by the following tuple:

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

2.5 PARTITION POINTS AND PARTITIONS

A change in harmony can only occur when the state changes. The state can change only when at least one note starts or ends. We call the moment of state change a *partition point*.

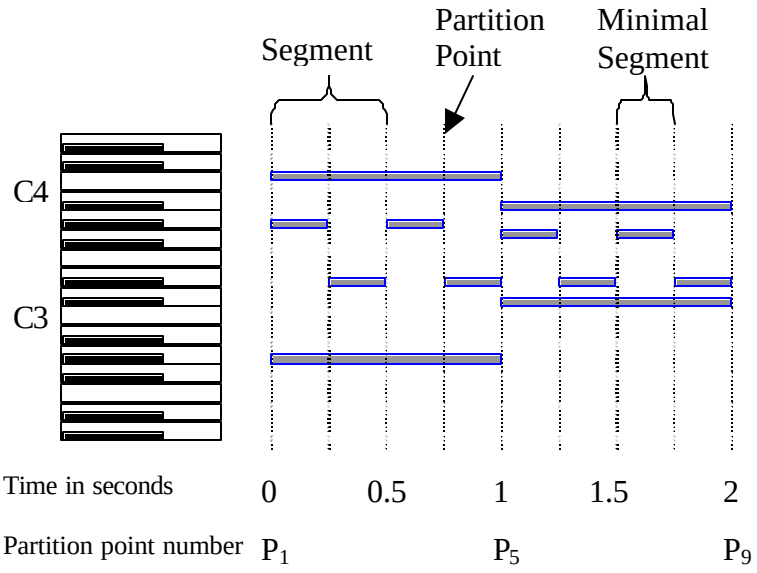

Figure 3 : Segments and Partition Points for $M_{beethoven1}$

The *set of all partition points*, P_{all} is the set of $start_n$ and end_n for all notes, *n*, in a performance, *M*, removing all duplicates. Thus, if several notes start or end at time *t*, only a single element with value *t* is entered into P_{all}. Moreover, the set of all partition points, P_{all}, defined over a performance, *M*, contains all points where the state changes.

For the performance $M_{beethoven1}$, P_{all} is the following:

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

P_{all} is derived from the start and end times of the notes in M and thus can have no more than twice the number of elements in M.

Lemma 2.5.1: $|P_{all}| \leq 2|M|$

Proof:

1. A partition point, by definition, happens only where at least one note begins or ends.

2. Each note has exactly one start and one end.

3. Consider the following two cases:

   a. If every note in M starts at a unique time, distinct from the start or end time of all other notes in M, and every note ends at a unique time, distinct from the start or end of all other notes in M, then $|P_{all}| = 2|M|$

   b. If the start or end of any note in M coincides with the start or end of any other note in M then $|P_{all}|$ is reduced, since P_{all} is the set of $start_n$ and end_n for all notes in M, removing all duplicates. Thus, $|P_{all}| < 2|M|$

¦

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.

Since each partition point, p_i, in P_{all} is unique (due to duplicate elimination), an ordering can be imposed on P_{all} by sorting its elements by value. Let the partition point with the earliest time be p_1 and the point with the latest time be $p_{|Pall|}$.

For example, 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.

$M_{beethoven1}$ has partition points every quarter of a second. This is strictly a result of the even tempo of the performance. If the performance tempo were to vary, then the length (in time) 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 4. Assume both a constant tempo of one quarter note per second and that each note begins the exact instant the previous note ends. In this case,

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

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

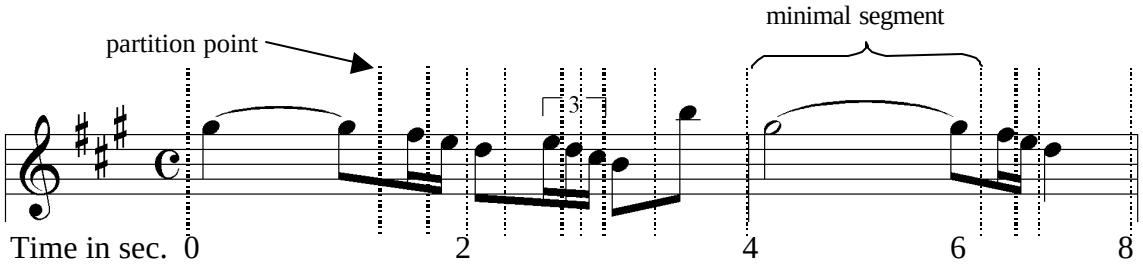

Figure 4 : Debussy, The Little Shepherd, mm 1-2

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

Since the elements of P_{all} are sorted, a partition 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 partition") or "1" (for "in partition") to each bit and the resulting binary number uniquely identifies a partition of a given performance. Figure 5 shows partitions 100100101 and 100010001 of $M_{beethoven1}$. Thick vertical lines indicate partition points that divide the music. Dotted lines are partition points that do not divide the music.

It is important to mention that there are performance gestures that may affect perceived harmony, such as dynamics and timbre. While the effects of these things are important to study, they lie outside the scope of this paper.

Every partition includes the first and last elements of P_{all}. This is necessary for proper formulation of segments and segmentations (defined in a later section). The first and last elements of P_{all} will always have their bits set to "1" and the bits that uniquely identify a partition are those for partition points p_2 through $p_{|Pall|-1}$. Thus, any partition can be uniquely identified with $|P_{all}| - 2$ bits. That said, we adopt the convention of identifying partitions using one bit for every partition point, including the first and last.

We now turn to the topic of *good* partitions. Except for simple cases, we expect that P_{all} will not be the final partition. For example, if the notes C,E,G are played in order with no intervening notes, P_{all} would have the corresponding binary number 1111. These notes, however, would be better considered a single harmony (C Major) with the partition 1001.

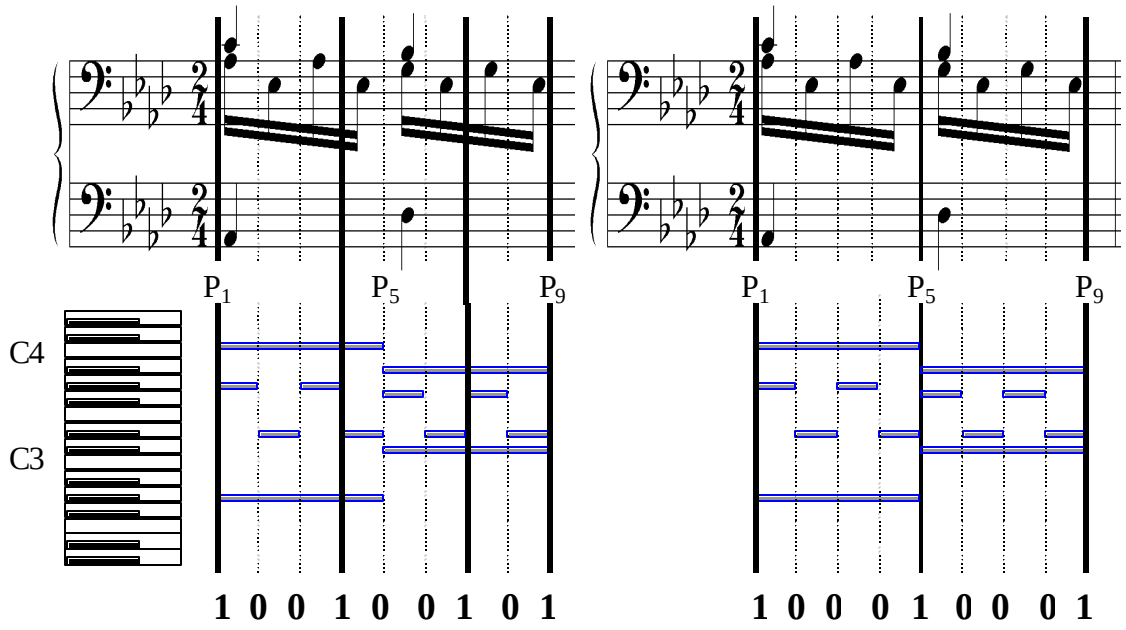

Figure 5: Two segmentations/partitions of $M_{beethoven1}$

A *good partition* consists of only the elements in P_{all} corresponding to harmonically significant changes in the state of the music. A harmonically significant change is the point where the chord name that a trained music analyst would assign to the current state changes. Partition 100010001 in Figure 5 is a good partition since it divides the measure into two segments, each of which corresponds to a single chord in the harmony of the piece. Finding a good partition for a piece of music is the problem of determining harmonically significant partition points.

2.6 SEGMENTS AND SEGMENTATIONS

We need to introduce a structure over partitions where we can define chords. We call this a *segmentation*, which is comprised of *segments.*

A *segment*, $s_{i,j}$, is the interval between partition points, p_i and p_j, i < j. We define $s_{i,j}$ as a duple, $< p_i, p_j >$. Correspondingly, a partition P over M yields a set of segments S, called a *segmentation* of M.

Partition 100100101 from Figure 5 contains the set of partition points $\{p_1, p_4, p_7, p_9\}$. This defines a segmentation $\{< p_1, p_4 >, < p_4, p_7 >, < p_7, p_9 >\}$.

A *minimal segment* is between two adjacent partition points in P_{all}: p_i and p_{i+1}. Figure 3 identifies a minimal segment, as does Figure 4. Since minimal segments go from partition point p_i to p_{i+1} they may be specified by a subscript indicating only the initial partition point. For example, segment s_3 is the segment from p_3 to p_4.

Recall that the state of the music at time *t*, $State_t$, is the set of pitches derived from the notes of *M* sounding at time *t*.

By definition, the state is constant throughout a minimal segment. Thus, we can speak of the state of a minimal segment *s*, which we denote $State_s$.

Two minimal segments, s_a and s_b, are *equivalent* iff $State_{Sa} = State_{Sb}$. This relation is expressed as $s_a \equiv s_b$

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 4 is an example of this.

A *rest* is a segment where no notes sound. All rests are minimal segments.

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

Any segment between partition points p_i and p_j in P_{all}, where $abs(j-i) > 1$, incorporates at least one change of state and can be decomposed into minimal segments.

Any partition derived from P_{all} is defined only for a specific performance, since P_{all} itself is defined for a specific performance. Suppose the Sonata Pathetique were played at 60 beats per minute (b.p.m.) in one performance, labeled M_a, and at 66 b.p.m. in another, labeled M_b. Since the start and end times of the notes in M_a and M_b differ, it may not be possible to directly compare segmentations or partitions of the two performances. For this we need to define an equivalence relation between performances.

Let s_i^k be the *ith* minimal segment in performance M_k. Let P_{all}^k be the set of all partition points for performance M_k. Two performances, M_a and M_b are *segment equivalent* when the following conditions hold:

1. $|P_{all}^a| = |P_{all}^b|$

2. $\forall i$ s.t. $1 \le i \le |P_{all}^a|$, $s_i^a \equiv s_i^b$

This simply states that two performances are equivalent if they are composed of equal numbers of minimal segments and that, for all *i*, the *i*th minimal segment in performance *a* is equivalent to the *i*th minimal segment in performance *b*.

Segment equivalence between two performances, M_a and M_b, is expressed as $M_a \equiv M_b$.

Note that segment equivalence is achieved by creating two performances with the same ordering of the same set of notes. Tempos may vary drastically as long as the ordering is

preserved. Segment equivalence is not preserved if M_a contains even one note not found in M_b, or if the ordering between notes varies between performances.

Segment equivalence may be applied to segments *within* a particular performance as well as between performances. For example, in $M_{beethoven1}$ the segments $s_{1,3}$ and $s_{3,5}$ are segment equivalent, since both conditions for segment equivalence hold between $s_{1,3}$ and $s_{3,5}$.

Theorem 2.6.1: If two performances, M_a and M_b are *segment equivalent*, then an algorithm for segmentation and labeling that uses only the state information found in the minimal segments will generate identical analyses for M_a and M_b

Proof:

1. If $M_a \equiv M_b$, then the only difference between them is in the absolute durations of their minimal segments.
2. An algorithm that uses only state information has no access to information about the durations of minimal segments
3. Thus, M_a and M_b are identical with respect to the algorithm and must generate identical analyses.

¦

Corollary 2.6.1 : If two performances, M_a and M_b are *segment equivalent*, then the chordal analysis for M_a may be applied M_b.

Proof: Follows directly from Theorem 2.6.1

¦

3 The Problem Size

Given a piece of music, M, and its set of partition points, P_{all}, the segmentation problem is to find a good partition, $P_{good} \subseteq P_{all}$, that contains only the partition points corresponding to harmonically significant changes in the music. We use the terms "segmentation" and "partition" interchangeably in this discussion, since each unique partition of a piece defines exactly one unique segmentation.

In monophonic music, no two notes sound at the same time. This can be expressed in terms of the states of a performance, *M*, at time *t*.

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

An unaccompanied melody is an example of monophonic music. Consider a very simple melody such as "Happy Birthday." The first two phrases of Happy Birthday are shown in Figure 6.This small excerpt contains 12 notes, giving a possible 24 partition points. This number is greatly reduced if one assumes that note *i* begins at the exact moment note *i*-1 ends This lowers the number of partition points to $|M|+1$, the smallest number any monophonic piece may have. This situation is shown in Figure 6.

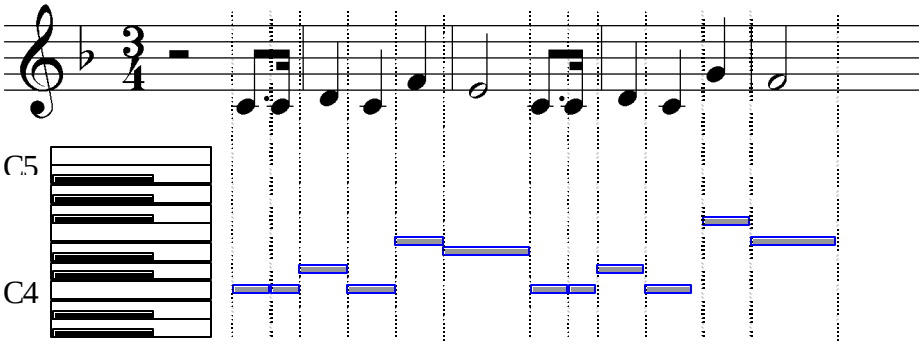

Figure 6 : Happy Birthday

Lemma 3.0.1: For a piece of monophonic music, whose notes are of non-zero duration, $(|M|+1) \leq |P_{all}|$.

Proof: Let *M* be a monophonic performance whose notes are ordered by start time.

1) A single-note performance has two partition points. Thus, $(|M|+1) = |P_{all}|$

2) If each note in *M* begins at the exact moment the previous note ends (i.e.,for all notes $i > 1$, $start_i = end_{i-1}$), then each note from 2 to $|M|$ contributes a single partition point. This is because end_{i-1} assures the entry of the partition point corresponding to $start_i$ into P_{all}. The first note contributes two partition points. Thus, $(|M|+1) = |P_{all}|$

3) Assume *M* to be in the state described in Step 2. If the start time of any note $i>1$ is advanced, it will start before note *i*-1 ends, making the piece non-monophonic.

4) Assume *M* to be in the state described in Step 2. If the start time of some note $i>1$ is delayed, it will start after note *i*-1 ends. This means that $start_i \neq end_{i-1}$ and note *i* will contribute two partition points to P_{all} while it adds only one note to *M*. Thus, $(|M|+1) < |P_{all}|$

This accounts for all cases. Thus, $(|M|+1) \leq |P_{all}|$

In a previous section, we showed that any segmentation for a particular performance can be uniquely identified by a binary number whose number of bits is equal to the number of partition points in $|P_{all}|$. We now use this result to formally state the size of the search space.

> Theorem 3.0.1: Given a performance M with the set of all partition points P_{all}, there are $2^{|Pall|-2}$ ways to segment M.
>
> Proof:
>
> 1. Each unique partition of a piece defines exactly one unique segmentation.
> 2. Every partition includes the first and last elements of P_{all}.
> 3. Each partition point, p_i, where $1 < i < |P_{all}|$, is either included in the partition (segments the music) or is not. If p_i is in the partition, we label it with "1." If p_i is not in the partition, we label it with "0."
> 4. This labeling process creates a unique binary number where bit i is the label for p_i.
> 5. Since only the partition points 2 through $|P_{all}|$-1 may vary, this binary number is of length $|P_{all}|$-2.
> 6. There are $2^{P_{all}-2}$ binary numbers of length $|P_{all}|$-2.
>
> ¦
>
> Corollary 3.0.1: For any monophonic performance, the number of possible segmentations falls between $2^{|M|-1}$ and $2^{2|M|-2}$.
>
> Proof: by Lemma 3.0.1, Lemma 2.5.1 and Theorem 3.0.1

As can be seen from this analysis, there are a prohibitively large number of ways to segment even a short monophonic melody. For example, there are at least $2^{11} = 2048$ ways to segment the 12-note example in Figure 6.

Not all music is monophonic and the number of different ways to segment a non-monophonic piece of music can be significantly lower than $2^{|M|-1}$. This is because the number of possible ways to segment 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; the extreme case is when all notes start and end at the same time. Such a situation is rare.

The Beethoven fragment $M_{beethoven1}$ is typical: the fragment has 12 notes, but only nine partition points, due to concurrent start and end times for many notes. This results in

many fewer ways to partition the fragment, but the number of partitions is still 2^7 for a single measure.

Of the $2^{|Pall|-2}$ ways some *M* can be partitioned, relatively few of these will correspond to how a well-trained musician would segment the piece on basis of the harmony. We previously defined a good segmentation as one that divides the music at points where a human expert would say the harmony changes.

The size of the segmentation space begs the question: "Does one really need to examine $2^{|Pall|-2}$ segmentations to find a good segmentation?" The size of the problem space can be greatly reduced by choosing a segment labeling and scoring method that generates scores that are *interval scaled, additive* and *transitive.* If we further constrain the system to label and score a segment *locally*, we will show that the segmentation problem can be solved in $O(|P_{all}|^2)$ steps.

The Roman-numeral notation for chord names labels a chord with respect to a key. Chord names are Roman numerals indicating the scale degree upon which the chord is built. For example, a G major triad is a "V" chord in the key of C since its root, G, is the fifth note in the C scale. The same triad is a "I" chord if the key is G.

The key of a particular segment of music is often impossible to determine without examining numerous segments. What's more, Roman numeral-style analysis often discards certain chords when determining key area, labeling them as non-structurally important. This makes "classical" harmonic notation inherently "global," since a final determination of harmonic label for a particular segment depends on analysis of other, often non-juxtaposed, segments to determine the key area.

This opens up the possibility that to label a segment, *s*, one would have to consider all possible segmentations containing the segment *s*. There are the $2^{|Pall|-2}$ segmentations of a given performance. Fixing a segment assures that at most two partition points, in addition to the initial and final ones, will be included in any segmentation. Thus, there are at least $2^{|Pall|-4}$ segmentations containing segment *s* that must be considered in determining a label for *s*. There is also a *grounding* issue. If every segment must consider all possible contexts before being labeled, how is the first segment labeled?

Knuth (Knuth and Plass 1981) formulates breaking a paragraph of words into optimal length lines in a manner similar to our formulation of the music-segmentation problem, mentioning that *n* breakpoints result in 2^n ways to segment a paragraph into lines. His solution depends on the fact that the ideal and maximum allowed line length is known in advance. This lets him limit the context to a relatively small number of breakpoints (usually no more than twenty) and thus constrain the combinatorial explosion. The grounding problem is also taken care of by the maximum allowed line length, forcing an initial break that allows the algorithm to proceed.

Unfortunately, one does not know the harmonic rhythm of a piece of music beforehand. It may be that the entire piece consists of a single chord or that it breaks into a thousand chords. The rate at which chords change may also vary drastically throughout the music.

Thus, we are unable to use a variant of Knuth's solution to constrain the complexity of our problem.

One approach to speed segment labeling and scoring is through enforcing the *constraint of locality*. The constraint of locality is met when every segment can be labeled without contextual information, e.g., information about other segments. This means that only a single segment (possibly composed of multiple minimal segments) need be considered in the course of generating a label.

One segment in a performance, *M*, may contain up to |*M*| notes. Recall that $|P_{all}| \leq 2|M|$. Assuming each note in a segment is considered in a single operation, it takes $O\left(\frac{|P_{all}|}{2}\right)$ operations to label a segment. This can be reduced to constant number of operations if the notes in *s* can be merged into a fixed number of pitch classes.

Note that this savings comes from not having to consider all possible segmentations containing *s*, the segment we wish to label. This could also be achieved by considering any *fixed* number of segments in the course of labeling *s*. This would allow some context consideration in the course of labeling a segment. The choice of how many segments and what their relationship to *s* should be is outside of the scope of this paper and we limit ourselves to labeling segments without context.

"Jazz-style" chordal notation uses absolute labels that do not depend on key area. Chords are identified by root pitch class and chord quality. Thus, Jazz notation is ideal for local labeling of segments. If one uses this notation and further assumes that all harmonies in a piece are important and should be labeled (as opposed to discarding some based on perceived functional insignificance), then segments may be locally labeled, without reference to context. This is the key simplifying assumption for the segmentation problem.

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 partition are subsets of S_{all} and thus no additional segments must be labeled once S_{all} is labeled.

S_{all} for *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 given below:

$$|S_{all}| = \binom{|P_{all}|}{2} < \frac{|P_{all}|^2}{2}$$

Let *c* be the number of possible chord labels. Given that $|S_{all}| < \frac{|P_{all}|^2}{2}$, labeling all possible segments involves making no more than $c\frac{|P_{all}|^2}{2}$ comparisons.

Once all possible segments (<p_i, p_j> where $i < j$) are labeled and placed in a lookup table, the size of the space to be searched when generating a labeled good partition is $O(|P_{all}|^2)$ We use this value as the size of the search space for the problem of finding a good partition of a piece of music when the constraint of locality applies.

4 MAPPING HARMONIC ANALYSIS TO A DAG

Given segments that can be labeled and scored *locally* and whose goodness-of-fit scores are *additive* and *transitive*, we can pose the segmentation problem as a search for the highest-reward path through a directed acyclic graph (DAG). This can be done as follows:

1. Create one vertex for each partition point.

2. Sort the vertices according to partition-point number.

3. Make an edge between each vertex, *i* and every other vertex *j*, where *i*<*j*.

4. Make the reward (cost) for each edge *i*,*j* equal to the score of the segment from partition point *i* to partition point *j*.

The result of these steps is a DAG, where each vertex represents a partition point and each edge represents one segment. The source vertex represents the initial partition point and the terminal vertex is the final partition point. Any path from the first vertex to the last one represents a segmentation of the piece.

Figure 7 shows a short passage of music represented as a DAG. Partition points (vertices) are labeled with Greek letters.Two different segmentations are represented on the graph. The path along the gray edges represents the segmentation whose scores are shown in gray. The path along the black edges corresponds to the segmentation scored in black.

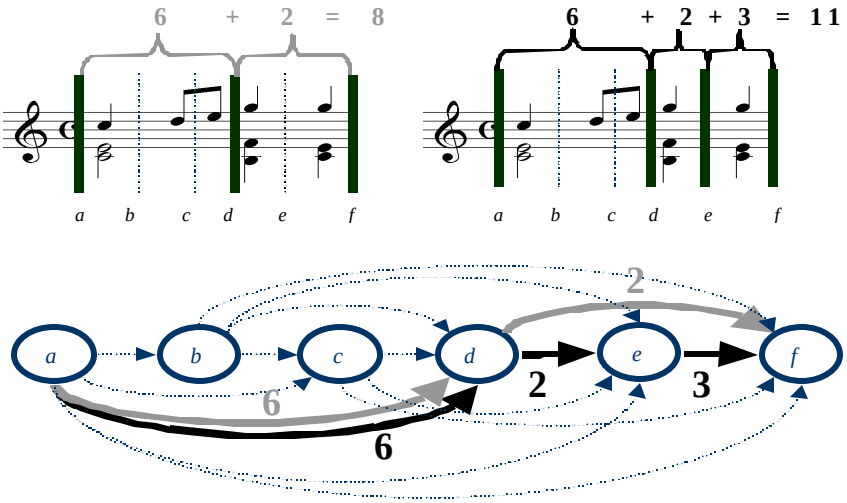

Figure 7: Two segmentations expressed as paths through a DAG.

Once the segmentation problem is cast as a search for the best path through a weighted DAG, standard graph search methods may be applied to find the best segmentation. In particular, the *relaxation* algorithm may be applied to find the best path (segmentation) in O(*E*) time, where *E* is the number of edges (Cormen, Leiserson et al. 1990).

Since *E* is equal to the number of possible segments, $|S_{all}|$, and $|S_{all}| < \frac{|P_{all}|^2}{2}$ this algorithm finds the optimal solution in $O(|P_{all}|^2)$ steps. Note that this savings is only possible when segments are scored locally and the scoring metric is transitive and additive. We believe that these are reasonable assumptions.

5 SEGMENT LABELING AND SCORING

To label segments, we need a way to determine the closest match between the set of chordal labels and a given set of notes. This label must also be scored in a way that is additive so that a path through the graph that contains *s* may add the reward for going through *s* to the total path reward in a meaningful way. This section discusses our labeling and scoring framework.

The chromatic scale and its associated 12 pitch classes form the basic set of items used to generate the structures for tonal 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. These tuples are positive displacements in the space of pitch classes in relation to a *root* pitch class, and they imply templates that describe musical structures. The templates are related to atonal set theory (Forte 1973), the chromagram (Wakefield 1999) and the work of Ulrich (Ulrich 1977).

An example template is the following: Given a root (pitch) class, *r*, the tuple <0,4,7> represents the pitch-class relations to *r* in a major triad. Letting *r* = 2, this results in a chord described 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 Templates

Number of Notes	Name	Preference Rating	Template representation
zero	rest	15	<>
one	single note	14	<0>
two	major 3rd / minor 6th	13	<0 4>
	minor 3rd / major 6th	12	<0 3>
	tritone	11	<0 6>
	fifth	10	<0 7>
three	major triad	9	<0 4 7>
	minor triad	8	<0 3 7>
	diminished triad	7	<0 3 6>
	augmented triad	6	<0 4 8>
four	maj-min (dom)7th	5	<0 4 7 10>
	half diminished	4	<0 3 6 10>
	fully diminished	3	<0 3 6 9>
	major 7th	2	<0 4 7 11>
	minor 7th	1	<0 3 7 10>

5.1 GENERATING THE MINIMAL SEGMENTS

The first step in processing *M* is generating P_{all} and a set, S_m, containing each minimal segment. Pseudo code for this is given in Figure 8. Note that the pseudo code assumes "well-formed" MIDI, where each "note on" is followed by a corresponding "note off."

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* currently sounding.

The function *GetNextNoteEvent* finds the next note event from the input MIDI stream by reading MIDI events as they occur, returning on the first "note on" or "note off" encountered. It then returns *deltaTime*, *keyNumber*, *velocity* and *eventType* for the event. If no note event is found, *getNoteEvent* returns "NULL."

```
FOR i = 0 to 11, state[i] := 0, END
time := 0
Pall[1] := <time,1>
i := 1
deltaTime,keyNumber, eventType := GetNextNoteEvent(MIDIfile)

WHILE (eventType != NULL)
        pitchClass := mod12(keyNumber)
        IF deltaTime > 0 THEN
                Sm[i] := <time, time+deltaTime, state>
                time := time+deltaTime
                i := i+1
                Pall[i] := <time,1>
        END
        IF eventType = "note on" THEN state[pitchClass] := state[pitchClass] + 1 END
        IF eventType = "note off" THEN state[pitchClass] := state[pitchClass] – 1 END
        deltaTime,keyNumber, eventType := GetNextNoteEvent(MIDIfile)
END
RETURN Sm, Pall
```

Figure 8: Generating S_m and P_{all}

The main while loop updates the state vector with the pitch class of each new note event, adding one to the appropriate element for a "note on," subtracting one for a "note off." When a non-zero *deltaTime* is encountered, this signals the start of a new minimal segment and the *state* is saved to S_m before updating it with the new note event.

The time required to generate P_{all} and S_m is linear with respect to the number of MIDI events in the input file. 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 bound on the complexity is O(|*M*|).

4.2 Segment Weight Vectors

Recall that our system represents $State_t$ as a vector of 12 integers representing pitch class, where the value of each element, e_i, is determined by the number of notes of pitch-class e_i sounding at time t. Looking at $M_{beethoven1}$ in Figure 2, the state of the first minimal segment, $<p_1, p_2>$ is the following:

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

The pitch *weight* vector of a segment $<i, j>$ is the sum of the state vectors of the minimal segments that comprise $<i, j>$.

$$\text{Weight}_{<i,j>} = \sum_{i}^{j-1} \text{State}_{<i,i+1>}$$

Thus, for a minimal segment, the weight vector is equal to the state vector.

- $\text{Weight}_{<p2,p3>} = \text{State}_{<p2,p3>} = [1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]$

An example non-minimal weight vector is that for the segment $<p_1, p_3>$.

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

The weight that results from this addition does not represent the number of notes of each pitch class sounding in the segment $<p_1, p_3>$. The held "C" in the example is represented by a value of two in $Weight_{<p1,p3>}$. Similarly, the element for "A flat" has the value three in $Weight_{<p1,p3>}$ 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 where it is present, and the sixteenth note "A flat" receives a point for being in $<p_2, p_3>$. Note that, although the low "A flat" is actually held for four minimal segments, it only has a value of two in the weight vector. This is because the segments under consideration only includes two minimal segments of the four where the "A flat" sounds.

The generation of the *Weight* vectors for the elements of S_{all}, the set of all possible segments for a piece, may be done in time proportional to $|S_{all}|$. This can be done by use of the algorithm in Figure 9, which assumes the state for each minimal segment has already been placed in a lookup table.

```
IF  (lookupTable[a,b] != NULL)
        weight = lookupTable[a,b]
ELSE
        weight = GetWeight(a,b-1) + GetWeight(b-1,b)
        lookupTable[a,b] = weight
END
RETURN weight
```

Figure 9: GetWeight(a,b) function

Calculating 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 method of calculating note weight, when combined with the right segment-scoring algorithm, ensures segment scores are additive in a reasonable way.

5.2 SEGMENT SCORING

The rules of tonal harmony are very well developed, but do not cover every conceivable combination of notes. For example, harmonies spread over time (i.e., non-block chords) often have melodic notes, ornamentation, notes serving other functions (e.g., leading tones), or are simply tonally ambiguous (by intent of the composer). Thus, chordal analysis of any complex piece of music requires tradeoffs among possible labels and segments. We reflect these tradeoffs in a scoring function used in HarmAn.

The segment's score is determined by measuring the distance of its *weight* vector from the nearest chordal template. HarmAn scores a pitch-class weight vector by comparing it

to all combinations of the root pitch class (0 through 11) and template (there are currently 15 in use), and selecting the highest score. A high score indicates a close match to a template. A low score indicates a large distance from *any* template. The highest score is returned as the score for the weight vector, and the template and root that generate the highest score determine the label.

The score for a weight vector, given a particular combination of root pitch class and template is determined as follows:

1. Adjust the template by adding the root pitch class to each element and then taking mod12 of the result.

2. Sum the weight elements whose index number matches an element of the template. Call this the positive evidence, *E*.

3. Sum the weight elements whose index number does not match an element of the template. Call this the negative evidence, *N*.

4. Sum the count of template elements that form the index number of a weight element whose value is 0. Call these the misses, *M*

The score is calculated by the following equation:

$$\text{score} = E - (N + M)$$

Consider the pitch-class weight vector for segment $<p_1, p_3>$ from $M_{beethoven1}$.

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

The template for a minor triad is <0, 3, 7>. Let our root pitch class be "F," or *5*. The score for $<p_1, p_3>$, given an "F" minor triad is calculated as follows:

1. $template = \text{mod } 12(\{0+5, 3+5, 7+5\}) = \{5,8,0\}$
2. $E = \sum_{i \in template} Weight(i) = 5$ since $Weight(5) = 0, Weight(8) = 3$ and $Weight(0) = 2$
3. $N = \sum_{i \notin template} Weight(i) = 1$ since the remaining vector elements sum to 1
4. $M = 1$ since $Weight(5) = 0$

$score = E - (N + M) = 4$

The label for a segment $<i,j>$ is determined by performing this calculation for each root-template combination and returning the high-scoring label and root.

$$Score(Weight_{<i,j>}) = \underset{\substack{template \in Templates \\ root \in Roots}}{\arg\max} (E - N - M)$$

If two templates generate the same score, they form an equivalence class and the winner may be selected using any tie-resolution rule.

In our current implementation of HarmAn, we resolve ties between templates by selecting the template whose *root* pitch class has a positive value in the weight vector. If this does not resolve the tie, we use preferences based on prior probability of occurrence derived from analysis of a large number of Bach chorales.

The tie-braking rules and the scoring method we described here reflect, to some degree, our preferences. In particular, we are interested in building the *simplest possible system* that still returns reasonable harmonic analyses. This simplicity is an advantage in that it allows the establishment of a baseline against which more complex scoring measures may be compared. To this end, we use the template, scoring and tie-breaking rules described above in all the empirical tests described in later sections of this paper. Others may have different preferences; it is simple to change the preferences and scoring method to encode different preferences. The only constraints are maintaining additivity and transitivity.

5.3 COMMENTS ON SEGMENT SCORING

We chose to make the segment-scoring method length-neutral. Consider the following examples.

Segment scores are additive and are *intuitively meaningful*. This is because segment weight is considered in the calculation of segment scores. Consider again the pitch-class weight vector for segment $<p_1, p_3>$ from $M_{beethoven1}$ (Figure 2).

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

The best scoring template for this segment is A flat major, or {8, 0, 3}, which returns a score of 6. Now consider the segment $<p_3, p_5>$, which has the exact same weight.

$$Weight_{<p3,p5>} = Weight_{<p1,p3>}$$

Minimal segments $<p_3, p_5>$ and $<p_1, p_3>$ are *segment equivalent* and the segment labeling algorithm return the label of A flat major and a score of 6 for each of the segments.

Now consider the segment $<p_1, p_5>$.

$$Weight_{<p1,p5>} = Weight_{<p1,p3>} + Weight_{<p3,p5>} = [4, 0, 0, 2, 0, 0, 0, 0, 6, 0, 0, 0]$$

The segment-scoring algorithm does as one would expect and returns a label of A flat major for $<p_1, p_5>$. The score returned is 12, exactly the sum of the score of the two sub segments.

Were weight (i.e., note length) not to factor directly into the scoring algorithm, it would be possible for the sum of the two sub segments scores to be higher (or lower) than the

score of the longer segment where, as in this case, the label returned is the same. This would introduce a bias towards search paths composed of more short segments. Similarly, if weight were given some multiplier, a bias would be introduced for paths consisting of fewer, longer segments.

6 HARMAN, A FASTER SEARCH ALGORITHM

We have shown how, in principle, it is possible to segment and label a performance with chord names in $O(|P_{all}|^2)$ time. This is not, however sufficiently fast for real-time processing on lengthy pieces of music.

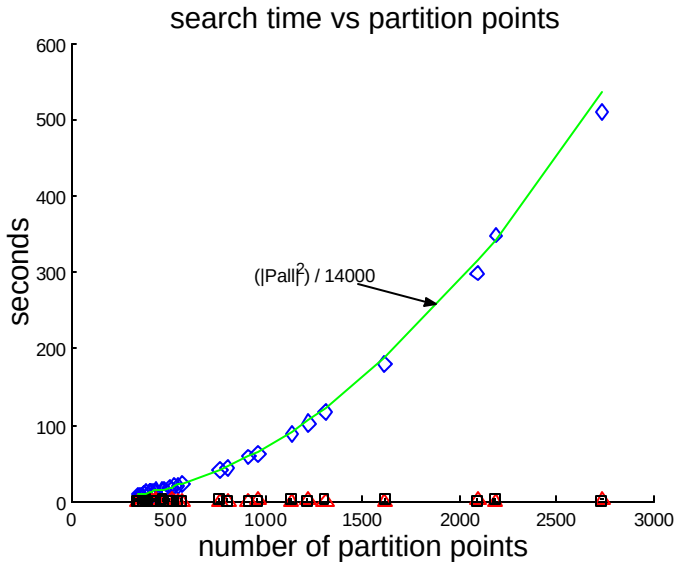

Figure 10: Segmentation times on 750 mHz Athlon with 128 mb memory

Figure 10 shows the result of running a relaxation-based search using our segment scoring method (described earlier). Diamonds represent segmentations done using relaxation. The computer used was a PC running Windows 98 Revision B with a 750 mHz Athlon processor and 128 megabytes of main memory. As expected, processing time increases quadratically with the number of partition points.

While time complexity $O(|P_{all}|^2)$ is much better than $O(2^{|P_{all}|})$, the relaxation algorithm is still not good enough to process music in real-time. For this, we need an approach that generates good segmentations in at most linear time. We have developed such an algorithm, which we use in HarmAn.

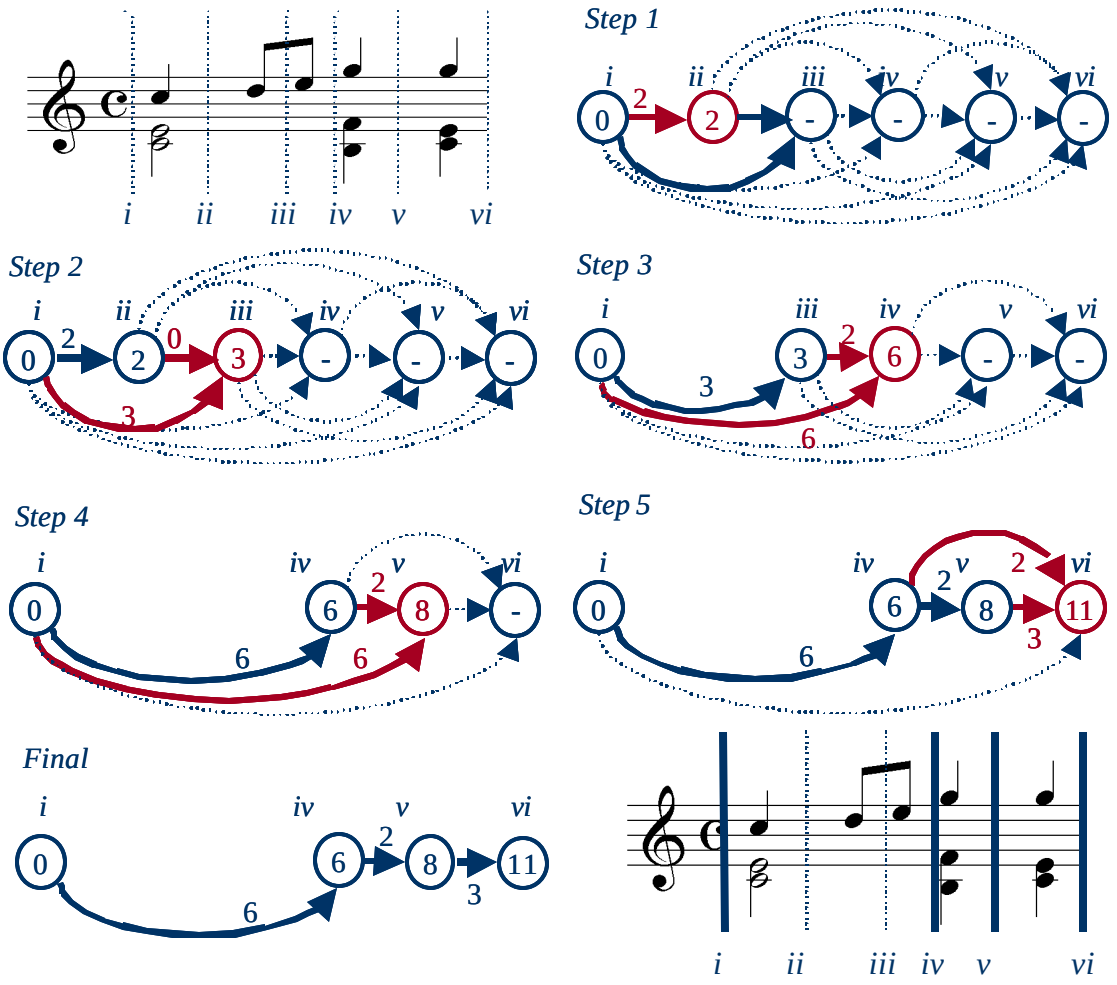

Figure 11: HarmAn search to find best segmentation

HarmAn uses a greedy approach to decide whether to include each vertex (partition point) in the path (segmentation). Segment scores and labels are determined as described in previous sections. HarmAn considers each vertex once, eliminating it if the path through that vertex does not appear to be a locally good one. If a vertex is eliminated from the path, all its outgoing edges are removed as well. This results in significant savings. In cases where the path through the vertex scores the same as the path skipping the vertex,that vertex is removed. This removes partition points in the music that separate segments that have the same chord label.

Figure 12 shows pseudo code for the search procedure. In this figure, $p(i)$ represents a partition point in P_{all}.

Figure 11 shows an example search through the DAG for a passage using the HarmAn search algorithm. Note that each node requires the system to score a maximum of two new edges. This allows the search to proceed in linear time with respect to $|P_{all}|$. Compare

this to the relaxation algorithm: the nodes appearing toward the end of the graph (i.e., closest to the "destination" node) will have, in the limit, in-degree of $|n|^2$, where n is the number of nodes in the graph, which in our case is the number of partition points. Thus, it is because we strictly limit the in-degree to a constant the time complexity of the HarmAn heuristic search is reduced.

The resulting complexity reduction is illustrated by the graph in Figure 10. Diamonds represent segmentation times for full search using the relaxation algorithm. All other markers represent variations on the HarmAn search algorithm. Note that the HarmAn algorithm never took more than two seconds to complete segmentation of a piece.

```
p(1).score = 0
p(1).prev =  null
p(2).prev = p(1)
p(2).score = score(GetWeight(1,2))

FOR a = 3 to (|P_all| )
        path1score =  score(GetWeight(a-1,a)) + p(a-1).score
        path2score =  score(GetWeight(a-2,a)) + p(a-2).score

        IF path1score > path2score
                p(a).prev = p(a-1)
                p(a).score = path1score
        ELSE
                p(a).prev = p(a-2)
                p(a).score = path2score
        END
END
```

Figure 12: HarmAn search

7 VOICE LEADING AND CHORD INVERSION

Once a piece is segmented, the template for each segment label is used to filter the original music. Notes matching a template element are considered harmonic. Those not matching the template are considered non-harmonic. HarmAn places harmonic and non-harmonic notes in separate MIDI tracks.

Harmonic notes, if extended to the length of a segment, may provide information about the voice leading of the piece of music. For each harmonic note, HarmAn outputs a note of the same absolute pitch lasting from the beginning to the end of the segment. The resulting sequence of notes shows voice-leading in a manner similar to a metric reduction. HarmAn places these in a separate MIDI track. HarmAn determines chord inversion by simply looking at the lowest note in the voice-leading track and finding the matching element in the template's segment.

Figure 13 shows an example of HarmAn's voice-leading output annotated with the segment chord labels on Bach's Two Part Invention #13 in A minor (chord inversion is not notated in this example). For this analysis, all templates in Table 2 were used, save

those for major and minor 7th chords. The search method used was the HarmAn heuristic, processing partition points start-to-finish.

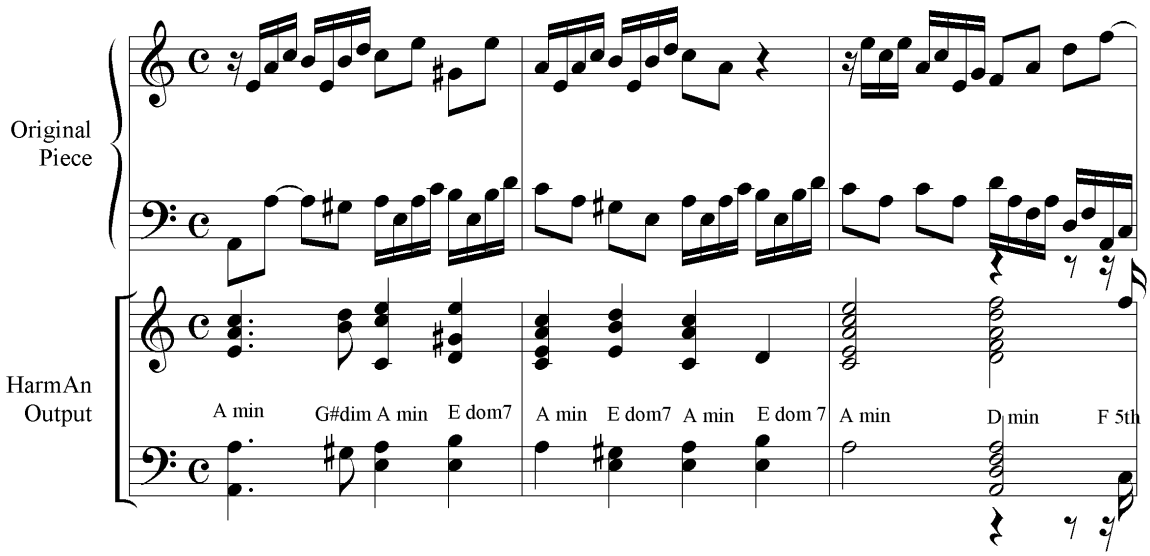

Figure 13: Bach 2 part invention no. 13 in A minor

8 EXPERIMENTAL EVALUATION OF HARMAN

In this section, we show the results of experimental evaluation of HarmAn. We first present an *intrinsic* evaluation, where we examine the solution quality of the HarmAn heuristic search compared with relaxation-based search. Here, we assume that relaxation-based search yields the optimal answer.

To determine how well HarmAn works, we also performed an *extrinsic* analysis, comparing its analysis to analysis in the literature. It is important to note that music analysis is, at times, open to different interpretations. Thus, while we may have an optimal algorithm or nearly optimal one and our scoring mechanism may be very good, different correct analyses may arise.

8.1 EVALUATION OF SEARCH ORDER

HarmAn (using either the heuristic or relax-based algorithm) processes music in a start-to-finish way with no backtracking. We justify this start-to-finish heuristic in the following way. Music 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 previously heard passages. The expectation for what comes next is determined by what has just been heard, and it is reasonable to assume that the set of likely segments under consideration by a human is greatly constrained by what has already transpired in the music. Thus, a greedy, start-to-finish approach is reasonable.

If a greedy, start-to-finish approach is reasonable, then the following suppositions should also be true.

1. Segmentations generated by HarmAn search should generate scores very close to those generated by relaxation search.

2. The order in which HarmAn considers the partition points (start-to-finish, finish-to-start, random, etc) should strongly affect the resulting segmentation.

3. Start-to-finish should be the best HarmAn search order through the partition points, and this should be reflected in higher-scoring segmentations.

In order to test these suppositions, we assembled a corpus of 32 pieces of Western Tonal piano music. These were Bach's 15 Three Part Inventions (Sinfonia), Seven Bagatelles from Beethoven's Opus 33 and Chopin Nocturnes 10 through 19. Each piece was segmented 43 times. One segmentation was generated by the relaxation search. One segmentation was generated by HarmAn processing the partition points in start-to-finish order. One segmentation was generated by HarmAn processing the partition points in finish-to-start (reverse) order and 40 segmentations were generated by HarmAn processing partition points in forty different random orders.

The score of each segmentation of a particular piece was recorded, and all scores for a particular piece were normalized to the high score on that piece so that comparisons could be made across different pieces. The results of the individual trials, broken down by piece and composer are shown in Figure 14 a, b and c. Figure 14 d shows individual trial normalized scores versus the number of partition points in the piece. In all of Figure 14, a diamond represents the score of relaxation search, a triangle represents HarmAn start-to-finish search, a square represents HarmAn finish-to-start and a point represents HarmAn random order search.

Figure 15 contains a box and whisker plot for each type of search. All trials for all 32 pieces in the corpus were included in the data for this figure. Each box has lines at the lower quartile, median, and upper quartile values. Median values for each search type are also identified by their numeric value. The whiskers are lines extending from each end of the box to show the extent of the rest of the data. Outliers are data with values beyond the ends of the whiskers. Outliers are indicated by the '+' sign.

Recall that since relaxation does a full search of the segmentation graph, its median score is 100% of the possible score. HarmAn search, when applied to the partition points in a start-to-finish (forward) manner achieves segmentation scores with a median 98.7% of the scores generated by relaxation search. Finish-to-start (backward) search achieves a median 98.2% of the maximal possible score. Neither forward nor backward search ever achieved a score below 95% of the maximal one. These findings support Supposition 1 and indicate that HarmAn search generates good segmentations when compared to full search using relaxation.

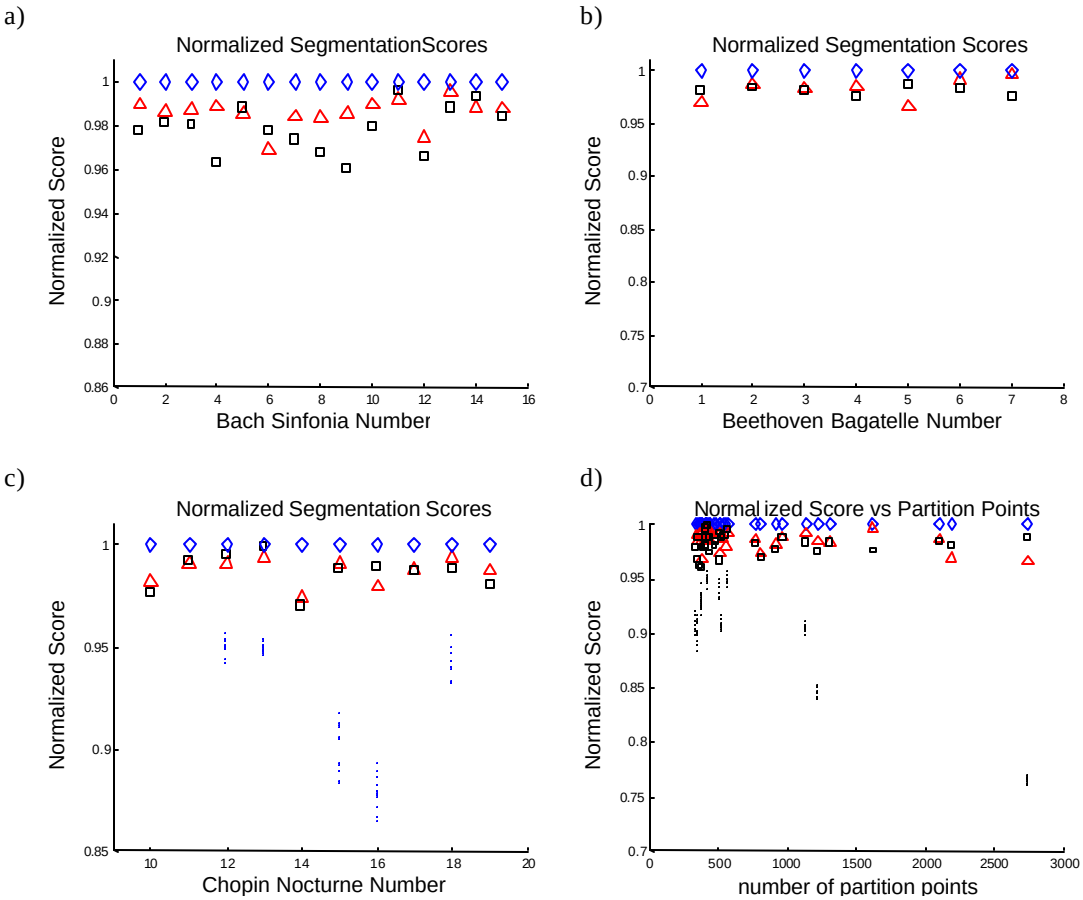

Figure 14: HarmAn search vs. Relaxation search

Using random-order selection of partition points for the HarmAn method generates significantly lower scores. Random search had a median score of 90.5% of the maximal. The lowest score generated by random order selection was on Beethoven Bagatelle no. 3, getting 72.2% of the maximal score. The distinctly lower scores achieved by random-order consideration of partition points support Supposition 2. The order in which the partition points are selected strongly influences the result of the search.

Supposition 3 was weakly supported by the data. While forward search did achieve the highest median score, it was only marginally better than backward search. What is more, reveals many instances where backward search outscored forward search.

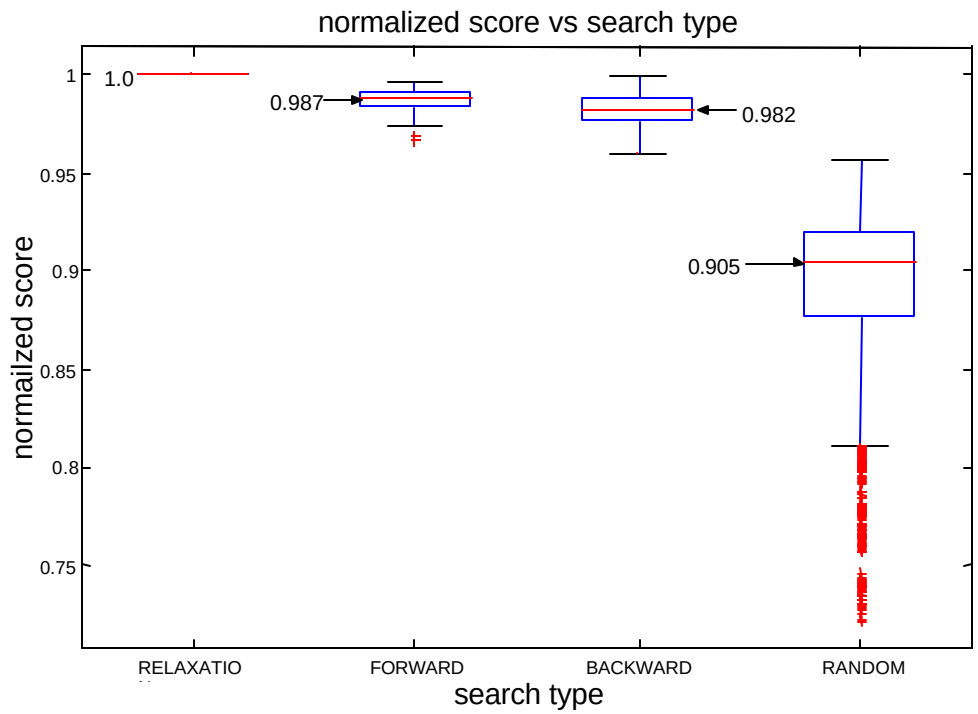

Figure 15 : Boxplot of normalized scores by search type

Start-to-finish search does have one distinct advantage over backward search: it allows a piece to be processed as it is being read (heard), making it the best choice for real-time applications, such as performing chordal analysis for the purpose of live accompaniment. Given this, forward processing remains the best choice when using the HarmAn search heuristic to do segmentation.

8.2 EXTRINSIC EVALUATION OF HARMAN

If it is true that tonal music was written to be analyzed in a start-to-finish way, our segment scoring algorithm is a good one, and that segments can be correctly labeled without regard for large-scale context, then HarmAn start-to-finish, greedy search 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.

Note that in the results against which we compare HarmAn, much of the analysis is in Roman-numeral notation. In those cases, we converted to Jazz-style notation by hand (this is straightforward and unambiguous). Furthermore, while these systems do a somewhat different analysis than HarmAn, the basic results (segmentation and segment

labeling) are comparable, as Roman-numeral analysis requires both segmentation and labeling.

8.2.1 WINOGRAD : SCHUBERT. DEUTSCHE TANZE, OP. 33, NO. 7

Winograd (Winograd 1968) approached chordal analysis using generative grammars. Winograd's work was successful in correctly labeling the harmonies using Roman-numeral notation in pieces of music composed of block chords, but did not address segmentation. His segmentation was performed by hand.

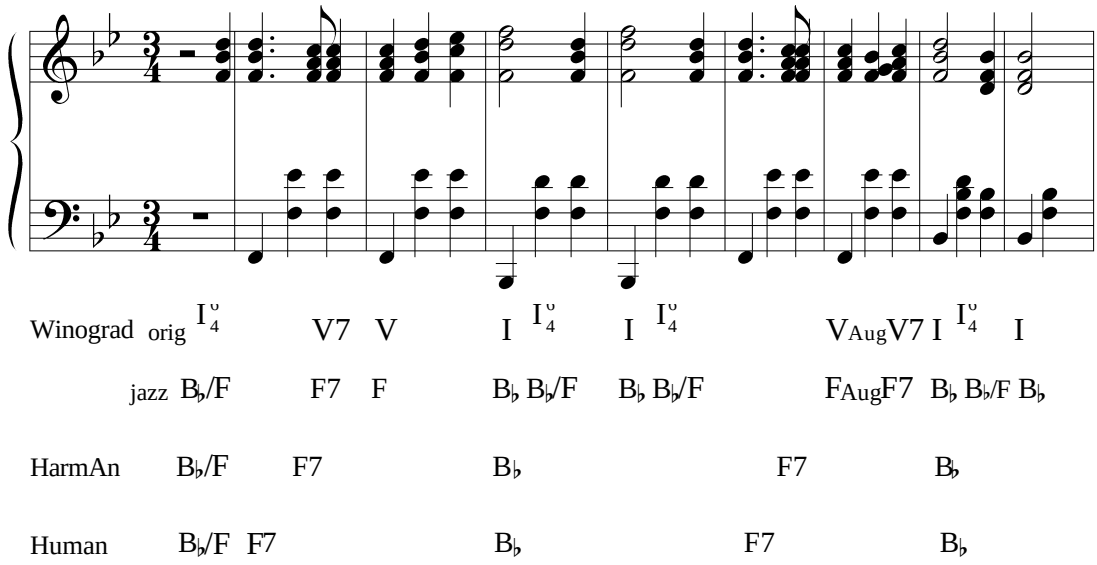

Figure 16: Schubert, Deutsche Tanze, Op. 33, no. 7

In order to test HarmAn on music with block chord figuration and to compare our results to Winograd's, the same pieces analyzed in his paper were analyzed by HarmAn. An example of HarmAn's analysis of the first eight measures of one piece is shown in Figure 16. In this figure, we show analyses by HarmAn, Winograd (in both Jazz and the original Roman-numeral notation) and a hand-analysis by the first author of this paper. Note that for all the examples of musical analysis in this paper, a chord name is placed at the beginning of the time span it covers. Thus, the F7 chord in the second measure continues through the third measure as well. Also, when a note other than the root of the chord is found in the bass, the Jazz style chord notation will place the chord name over the bass note name. Thus, $\frac{A\text{-}}{C}$ or A-/C indicates an A minor chord with C in the bass.

The results generated by HarmAn for this passage generally agree with Winograd's analysis. The major chord changes identified are the same. HarmAn unifies repeated chords of the same root, Winograd's system labels each one with its own inversion. This is a stylistic difference, rather than an actual disagreement. The only major disagreement is the placement of the dominant "F" chord in the sixth measure. == Ideally, this chord should begin at the start of the sixth measure, taking the B flat chord outlined on the first

beat of the measure as a suspension of the previous harmony. HarmAn is not designed to deal with suspensions and simply assumes the harmony of the previous measure continue until the full F7 is presented. Winograd's system waits until the start of the next full measure to label the chord as dominant. Interestingly, the second measure presents a similar situation and here Winograd's system agrees with HarmAn.

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

Temperley and Sleator's (Temperley and Sleator 1999) analysis begins by first performing beat finding in a manner strongly reminiscent of Lerdahl and Jackendoff (Lerdahl and Jackendoff 1983). These beats are then used in partitioning the piece into time spans, which are labeled as likely chords. The exact method used to determine the initial partition 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 preferring the same root name in successive chords, if possible, and preferring root names related by a fifth (seven half-steps), 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.

Figure 17 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.). In this figure we do not report chord inversion. As can be seen from the figure, with the exception of the fifth measure, Temperley's system successfully captured the correct chord roots, and HarmAn correctly labeled both root and chord quality.

In measure five, Temperley's system finds four chord roots, namely G, B flat, E flat and A flat. HarmAn reports only two. We believe that the two chosen by HarmAn are the correct ones for this measure.

While HarmAn came closer than the other system to correctly labeling the harmony in measure five, this measure also illustrates a weakness of the HarmAn approach. The notes on the start of the second beat of the fifth measure fit equally well in the Gm7(flat 5) chord and the E flat 7 chord. HarmAn processes with no notion of harmonic rhythm and thus has no way to disambiguate this case in an intelligent manner. It defaults to continuing the existing chord and places the E flat 7 chord one sixteenth note after the start of the beat, when the first note not fitting the Gm7(flat 5) chord is encountered. A more sophisticated system (such as a human analyst) would place the E flat 7 at the start of the second beat.

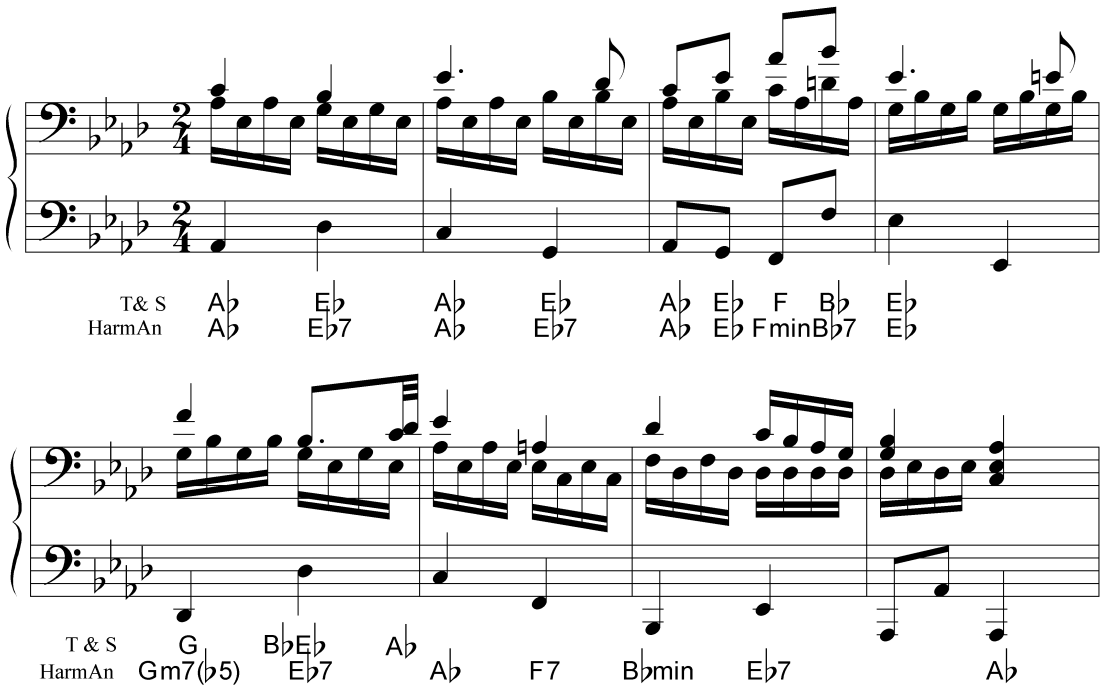

Figure 17: Beethoven, Sonata Pathetique, 2nd Movement

An ambiguous situation is encountered in the final measure. In this measure, HarmAn continues the E flat 7 until the second beat. Some analysts would interpret the entire measure as an A flat chord with a suspension of the E flat chord over the A flat bass. HarmAn allows only a single chordal explanation for any given segment and chooses the one supported by the majority of the notes, namely E flat 7.

8.2.2 MAXWELL : J. S. BACH, 1ST FRENCH SUITE, D MINOR SARABANDE

Maxwell (Maxwell 1992) built a system that performed chordal 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."

One strength of Maxwell's work is that it performed segmentation. Another strength is that it explicitly considers meter in determining chord placement. The metrical placement of notes is, however, given as input rather than inferred as in the work of Temperley and Sleator. Unfortunately, the paper describing the work lists only a small subset of the preference rules, and is unclear on what control structure mediates among these rules. Maxwell did not explicitly address the issue of the computational complexity of the segmentation problem, nor what portion of the space was actually searched.

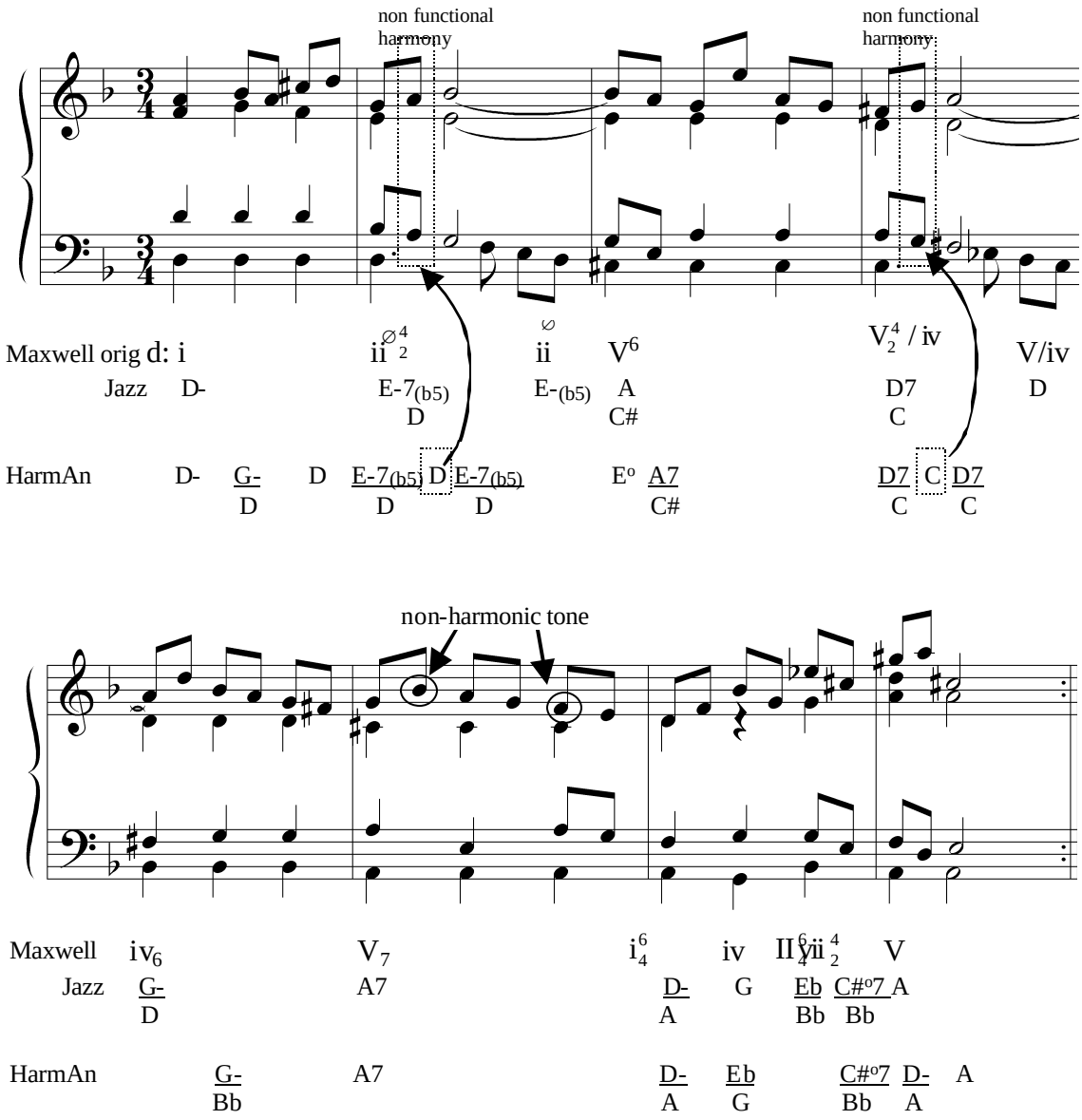

Figure 18: Bach, 1st French Suite, D minor Sarabande

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 18. Maxwell analyzed this piece because "it contains an abundance of non-harmonic activity without complex embellishments or passage-work" and "Its texture is considerably more complex than that of a chorale, especially with regard to harmonic rhythm". This passage contains numerous passing tones that are not harmonically supported (e.g., they are not consonant to the chord) and other tones that spell out vertical chords that are a side effect of contrapuntal (melodic) motion and do not

constitute functional harmonies. Maxwell's system was designed to reject both dissonant melody notes and non-functional harmonies in its analysis. HarmAn has no notion of harmonic function and rejects only dissonant notes while still labeling non-functional harmonies. This can be seen in the analyses generated by the two systems.

Measures two and four in Figure 18 show examples of harmonic by-products of contrapuntal motion in two voices. In both cases, HarmAn labels the harmony, correctly, and Maxwell's system refrains from labeling them. This is also the case in the more ambiguous situation found in the first measure. Here, the passing chord is a G minor (G-) and happens on the beat. The G- has a fifth relation to the D- chord on beat one. Both these things make the chord a much better candidate for a structural harmony than the D chord HarmAn labels in the second measure.

Measure six has passing notes in the top voice which are not made consonant by any passing notes in a lower voice and both systems correctly ignore these non-harmonic tones to label the entire measure as an A7 chord.

Overall, it seems the Maxwell system does slightly better than HarmAn in that it refrains from labeling non-functional chords arising from passing motion in several voices. However, Maxwell's system is significantly more complex and has metric (beat) information provided to the system as input. Taken in this light, HarmAn performs quite well.

9 CONCLUSIONS

The work described in this paper provides a framework upon which to build a system for chordal analysis of music. We established that, in the worst case, the partitioning and labeling of harmonies in tonal music is $O(2^N)$, where N is number of partition points in a piece. When the "constraint of locality" applies, and a segment scoring metric can be found that is both additive and transitive, the problem becomes $O(N^2)$.

Under these constraints, the problem can be cast as finding the best path through a single-source DAG, which can be solved in O(E + V) time. This translates to a worst-case time complexity of $O(N^2)$ for our problem, where N is the number of partition points. We then showed that the results of the $O(N^2)$ search can be closely approximated through the use of a heuristic, the "HarmAn heuristic," which allows $O(N)$ time search.

HarmAn is a software system based upon this framework, and it provides a good analysis of the harmonic structures in a typical piece of tonal music. HarmAn does this by exploiting the constraint of locality and a scoring function that is additive and transitive, and a relatively small set of rules for breaking ties. HarmAn does not require an understanding of tonal context nor any metrical information. The results achieved by this system compare well to those achieved by far more complex systems reported in the literature.

In those areas where HarmAn generates suspect results, it appears that additional context information could be used to improve the system's performance. For example,

information about meter and dynamics may give better hints for segmentation. In addition, knowledge about the harmonic structure of a piece of music, such as the types of scales used, would help improve the performance of the scoring algorithm.

10 REFERENCES

1. Cope, D. (1991). "Recombinant Music: Using the Computer to Explore Musical Style." Computer(July 1991).

2. Cormen, T., C. Leiserson, et al. (1990). Introduction to Algorithms. Cambridge, Massachusetts, MIT Press.

3. Dannenberg, R. B. (1993). Music Understanding By Computer. IAKTA/LIST International Workshop on Knowledge Technology in the Arts Proceedings, Osaka, Japan: Laboratories of Image Information Science and Technology.

4. Dannenberg, R. B. (1998). New Techniques for Enhanced Quality of Computer Accompaniment. Proceedings of the International Computer Music Conference, Ann Arbor, MI.

5. Ebcioglu, K. (1992). An Expert System for Harmonizing Chorales in the Style of J. S. Bach. Understanding Music With AI:Perspectives on Music Cognition. M. Balaban, K. Ebcioglu and O. Laske. Cambridge, Mass, MIT Press**:** 294-333.

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

7. Fujishima, T. (1999). Realtime Chord Recognition of Musical Sound: A System Using Common Lisp Music. International Computer Music Conference, Beijing.

8. Heijink, H., P. Desain, et al. (2000). "Make Me a Match: An Evaluation of Different Approaches to Score-Performance Matching." Computer Music Journal **24**(1): 43-46.

9. Hoffman, T. and W. Birmingham (1999). Samuel Scheidt goes MAD: a Multi-Attribute Domain CSP Approach to Harmonic Analysis. Ann Arbor, University of Michigan.

10. Horowitz, D. (1995). Representing Musical Knowledge in a Jazz Improvisation System. IJCAI-95 Workshop on Artificial Intelligence and Music.

11. Johnson, M. (1991). "Toward an Expert System for Expressive Musical Performance." Computer(July 1991).

12. Katayose, H. and S. Inokuchi (1993). "Learning Performance Rules in a Music Interpretation System." Computers and the Humanities **27**: 31-40.

13. Knuth, D. and M. Plass (1981). "Breaking Paragraphs into Lines." Software Practice and Experience **11**: 1119-1184.

14. Kraut, R. (1992). On the Possibility of a Determinate Semantics for Music. Cognitive Bases of Musical Communication. M. R. Jones and S. Holleran, American Psychological Association.

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

16. Linster, C. (1992). On Analyzing and Representing Musical Rhythm. Understanding Music With AI:Perspectives on Music Cognition. M. Balaban, K. Ebcioglu and O. Laske. Cambridge, Mass, MIT Press.

17. Marsella, S. and C. Schmidt (1992). On the Use of Problem Reduction Search for Automated Music Composition. Understanding Music With AI:Perspectives on Music Cognition. M. Balaban, K. Ebcioglu and O. Laske. Cambridge, Mass, MIT Press: 238-256.

18. Maxwell, J. H. (1992). An Expert System for Harmonizing Analysis of Tonal Music. Understanding Music with AI: Perspectives on Music Cognition: 335-353.

19. Moorer, J. A. (1975). On the Segmentation and Analysis of Continuous Musical Sound by Digital Computer. Computer Science. Stanford, CA, Stanford University.

20. Mozer, M. C. (1991). Connectionist Music Composition Based on Melodic, Stylistic, and Psychophysical Constraints. D. Gareth Loy. P. M. Todd. Cambridge, Mass, MIT Press.

21. Polito, J., J. M. Daida, et al. (1997). Musica ex Machina: Composing 16th-Century Counter-point with Genetic Programming and Symbiosis. Evolutionary Programming VI: Proceedings of the Sixth Annual Conference on Evolutionary Programming, 1997.

22. Scarborough, D. L., B. O. Miller, et al. (1991). Connectionist Models for Tonal Analysis. D. Gareth Loy. P. M. Todd. Cambridge, Mass, MIT Press.

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

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

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

26. Todd, N. P. M. (1992). "The dynamics of dynamics: A model of musical expression." Journal of the Acoustical Society of America **91**(6): 3540-3550.

27. Todd, P. M. (1991). A Connectionist Approach to Algorithmic Composition. D. Gareth Loy. P. M. Todd. Cambridge, Mass, MIT Press.

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

29. Wakefield, G. H. (1999). Mathematical Representation of Joint Time-Chroma Distributions. The Intl. Symp. on Opt. Sci., Eng., and Instr., SPIE'99, Denver, Colorado, USA.

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

31. Windsor, W. L. and E. F. Clarke (1997). "Expressive Timing and Dynamics in Real and Artificial Musical Performances: Using an Algorithm as an Analytical Tool." Music Perception **15**(2): 127-152.

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

33. Zimmermann, D. (1995). Exploiting Models of Musical Structure for Automatic Intention-Based Composition of Background Music. IJCAI-95 Workshop on Artificial Intelligence and Music.