Journal of New Music Research
2009, Vol. 38, No. 2, pp. 171–181

Routledge
Taylor & Francis Group

A System for Identifying Common Melodic Phrases in the Masses of Palestrina

Ian Knopke[1] and Frauke Jürgensen[2]

[1]Aspen Computing Solutions, UK; [2]University of Aberdeen, UK

Abstract

An important direction in Renaissance music research is the relationship of repetition to structure and compositional practice. This calls for very time-consuming analysis, and is generally unfeasible when applied to large collections of compositions. However, such large-scale analyses are necessary to allow us to identify trends across entire repertories. We describe a system based on the use of suffix arrays, which allows us to find instances of melodic repetition in an enormous body of music, within a reasonable amount of processing time. This system identifies all transpositions, inversions, retrogrades, and retrograde inversions of unknown melodic segments. It has been applied to the entire collection of masses by Palestrina, a corpus of over seven hundred mass sections and approximately one million notes.

1. Introduction

The rules of counterpoint can be viewed as a system for governing parallel interactions between two or more melodies. Traditional music theory tells us much about the details of writing such counterpoint, and provides us with rules for voice leading, cadential construction, dissonance treatment, and lists of common contrapuntal combinations. While these rules convey fairly robust solutions at a local level, they are of limited guidance towards the construction of entire musical compositions.

In general, the interaction between contrapuntal technique and the structural details of compositions has only recently been systematically explored, and only on a fairly small scale. Of even greater interest, attempts to answer these structural questions across entire corpuses of contrapuntal music, perhaps even leading to considerations of concepts such as style, have not generally been undertaken in a convincing manner.

Of course, there are practical difficulties with this type of study using traditional means. Much music theory work is still done by researchers through direct examination of individual scores, and investigations of entire collections require considerable amounts of time and effort on the part of researchers, in order to gather sufficient and convincing evidence to support hypotheses.

The methodology described here is the first part of a large-scale project that aims to undertake these sorts of large-scale, in-depth studies, by automating the analysis process of identifying contrapuntal melodies, modules and their transformations, beginning with single melodic lines. In brief, the system described here is capable of discovering all transpositions, inversions, retrogrades, and retrograde inversions of unknown melodies across a large corpus of compositions consisting of approximately a million notes, in a reasonable amount of time.

Note, however, that we do not claim to be able to solve the general question of how to handle these problems for all musical situations. These techniques are adapted to work specifically with the sixteenth-century contrapuntal material that we are working with. Some characteristics of this music allow us to make assumptions that may not apply especially well to other musical situations, and these have been addressed in the implementation.

Nevertheless, applying these techniques to a particular musical collection has proven instructive, and we hope in the future to be able to generalize these techniques to other musical situations.

This article is arranged as follows. The first section gives an overview of the Palestrina Masses, followed by an overview of the basic details of our system. The next

Correspondence: Ian Knopke, Aspen Computing Solutions, Unit 4A, Arena Design Center, Ashfield Road, London, N4 1FF, UK. E-mail: ian.knopke@gmail.com

DOI: 10.1080/09298210903288329 © 2009 Taylor & Francis

section explains the remapping technique we use that makes it possible for us to reuse some existing string-matching technologies with a music encoding scheme, followed by some information about phrase procedures and metadata storage. This is followed by details of our procedure for finding unknown, commonly reoccurring phrases using various string-matching techniques. Finally, some example matches are shown, as well as some overall properties of the entire mass set. We conclude with a discussion of the future directions of this project.

2. Palestrina's masses

Our collection for these experiments consists of 101 masses composed by Palestrina. Palestrina's first book of masses was published in 1554 (the first such volume published by a native Italian composer), and the last were issued after his death in 1594. A possible date of composition is, with some exceptions, very difficult to determine. Each mass consists of the various movements of the Mass Ordinary (Kyrie, Gloria, Credo, Sanctus, Benedictus, Agnus Dei), which in turn are usually subdivided into clear sections determined by divisions in the text. These sections tend to be marked by a change in texture, and often a change in the number of voices. The masses vary in the overall number of voices from four to six, with some sections in three voices.

The masses can be grouped roughly according to their musical source material; for example, into freely-composed masses, masses using a single melody as a source (such as a piece of plainchant), and masses borrowing another polyphonic piece such as a motet or madrigal. Each of these techniques brings with it certain compositional restrictions that will predispose the piece to behave in certain ways. For example, an imitative structure is more difficult to build over a cantus firmus in long notes, such as that in the very first mass, *Ecce sacerdos magnus*.

There are several reasons why the works of Palestrina make an excellent test collection for our methods. First, Palestrina's music has long been regarded as the model for an idealized Renaissance style, a tradition going back to theorists of the sixteenth century. At many universities, modern composers are still taught to write 'Palestrina-style' counterpoint in the Renaissance style as part of their basic training. Palestrina also wrote more Masses than any other major Renaissance composer. This gives us an enormous body of music with a standardized text that is stylistically consistent. A final reason is simply the availability of the collection in an encoded format.[1]

[1]Palestrina's masses were made available in electronic format (Humdrum encoding) in 2004 by Bret Aarden, currently at the University of Massachusetts—Amherst. This electronic edition, originally produced by John Miller in 1992, is based on the Casimiri edition of Palestrina's complete works (Palestrina 1981).

There are 1706,027 tokens in the overall data set; however, only 801,779 of these are actually notes, with an additional 92,895 rests. A breakdown of the collection by mass parts is shown in Table 1.

3. Incorporating musicological knowledge

Since we are aiming to make a system specifically to be applied to Renaissance counterpoint, we can make certain assumptions based on our musicological understanding of the repertoire in order to streamline our approach. Primarily, these assumptions allow us to reduce the number of tokens needed for representation. Certain rhythmic values, for example, never occur in these pieces. The shortest note value is an eighth note, and there are very few tuplets: the third Agnus of *Missa Ecce sacerdos magnus* has two quarter notes among half-note triplets, and the Kyrie of *Missa L'homme armé (1570)* and the Benedictus of *Missa Virtute magna* also have half-note triplets. Similarly, the available pitch space is limited by the vocal range in use at the time, and by the rarity of accidentals far outside the gamut. The only accidentals used are B#, E#, F#, C#, and G#. In addition, the rarity of large melodic leaps allows us to treat pitches as octave-equivalent without the risk of too many false matches: with the exception of octave leaps, leaps exceeding a fifth are rare.

Another set of assumptions concerns phrase segmentation. The main purpose of phrase segmentation in our approach is to limit the length of the strings that need to be compared for matches. In Renaissance vocal music, probably the most obvious approach to phrase segmentation would be by using text phrases or words, since text and musical phrases usually coincide. Unfortunately, the kern encoding of Palestrina's masses does not include text. Instead, we have chosen to use rests as a marker to determine phrase boundaries. A hand count of phrase boundaries within all movements of *Missa Brevis* and five randomly-chosen Kyries revealed that in most cases, phrases were bounded by rests: for example in the Kyrie of *Missa Brevis*, 39 phrases were counted, of which 35 ended with a rest. In the rest of cases, two phrases were

Table 1. Number of mass parts in our collection.

Mass part	Count
Kyrie	157
Credo	399
Benedictus	162
Agnus	191
Gloria	207
Sanctus	199

joined together without being separated by a rest. In no case was a phrase disrupted by a rest. While we have not comprehensively evaluated every mass, throughout the course of this research we have yet to encounter a case where the rule of rests-as-phrase boundaries is not true. This suggests that rests are a reliable positive indicator for a phrase boundary in this music. Exceptions to this rule should be easily spotted by the analysing musicologist, as a matched melody interrupted by a rest will simply present as two shorter matched melodies. In certain earlier or instrumental repertories, where for example the replacement of the dot of a dotted note with a rest is fairly common, this approach would need to be refined.

4. System overview

The basic workflow of our system is described in Figure 1.

Initially, a mapping of existing multi-character music symbols is made into single-character tokens. This can be created dynamically by constructing a list of all tokens in the entire set, classifying each token into a category, and remapping each category to a single alphanumeric character. Alternately, pre-existing mappings may also be used if the properties of the mapping are known ahead of time, such as with interval classes or single-octave pitch classes. The salient point here is that the representation must maintain enough complexity to model the musical processes we are interested in studying, while still mapping well to a smaller set of single-character tokens that can be used with string-matching techniques. All data extraction is accomplished using the *PerlHumdrum* toolkit for symbolic music processing (Knopke, 2008).

Each mass is then considered in turn, and each voice of each mass is cut into phrases. One aspect of this music is that rests almost always indicate phrase boundaries, and melodies here are not generally considered to cross phrase boundaries. In general, this assumption does not hold for most other types of music, and this greatly simplifies many of our analytical tasks.

Phrases are then separated into individual *Kern* tokens. Each token is then encoded using the previously created mapping and inserted into a *suffix array* structure, along with the remainder of its phrase. Each note is also inserted into a text database that stores all of the accompanying information about the note, including mass number, movement number, stave, and note index within that stave.

Once the entire collection of masses has been encoded in this way, the suffix array is sorted, aligning each common phrase contiguously. Every pair of lines across the entire suffix array is then compared for common tokens, and all matches are inserted by length into a priority queue. A list of the longest matches is produced and subjected to some additional filtering to remove spurious results. The final set of best matches is then converted back into *Kern* format, and optionally, into musical notation.

Specific aspects of some parts of the system are discussed in the following sections.

5. Representational issues

Pattern-matching techniques such as *suffix tries* and *suffix arrays* have their origins in string-processing and are often used in areas such as text editing, data mining and bioinformatics. Ostensibly these would seem to apply equally well to sequences of musical notes. However, the true situation is not so simple. Such algorithms tend to produce limited results when applied directly, and some sort of adaptation is almost always required.

5.1 Remapping music

One problem comes from the representation of musical symbols on a computer. Most string processing tools are designed to work with data tokens that consist of single alphanumeric characters, such as the ASCII character set, where each symbol is encoded as a single-byte character. However, most representations of music, including Humdrum's *Kern* format, involve multiple characters per token or symbol. For instance, a middle c quarter note would be represented in a Humdrum *Kern* score as the two-character token c4. A token such as 4.b-, representing a Bb dotted quarter note that is tied to another, unindicated value, requires four characters. More complex tokens are easily encountered. Because most string processing tools interchangeably equate the

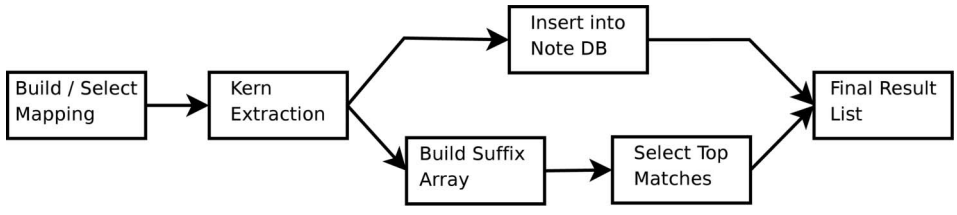

Fig. 1. Processing diagram.

notions of token and character, this creates extra difficulties of a practical nature that most text or bioinformatics applications never have to overcome.

One possible solution would be to reimplement the existing algorithms in special 'music' versions that have the ability to directly manipulate the musical symbols involved. For instance, one could write a sort algorithm that understands that c and c# are single tokens that sometimes consist of multiple characters, and can handle processing tasks accordingly. However, reimplementing existing programs has the potential to be extremely time-consuming.

The simple technique presented here allows us to make use of a vast amount of existing string processing technology unmodified. The approach we have taken in this problem is to remap existing multiple character symbols into single-token ASCII characters. Manipulations and searches are then performed using efficient string-mapping tools, and the resulting answers can then be reverse-mapped back to the original musical meanings as necessary.

An example mapping is given in Table 2, where each of the 12 pitches, including those with accidentals has been mapped to a single ASCII character. An example of this in use would be searching for the presence of f# minor arpeggios in a melodic sequence of pitches such as that in Table 3, using the mapping shown in Table 2. The remapped character string lfdlelhdjfdlhdjfm is much more compact. Because the token length is previously known (one character) there is no need for interstitial spaces or other delimiting characters. The chordal 'query' in this case would also need to be remapped from the pitches f# a c# to fdl.

This search operation can now be undertaken using many common computing tools, including tools such as grep or awk, commonly used in the UNIX environment for efficiently searching large bodies of text. As an example, using the Perl language, the query can be handled by something as simple as the following regular expression: $ans = ~/fdl/;, indicating by the return value whether that arpeggio would be found in the group of pitches.

In the case of Table 2 the actual mapping was built dynamically, in the order the pitches were first encountered in the set of scores. It is also possible to pre-assign a mapping if the set of tokens are already known, as in this case, although this may not always be desirable. Note that the actual characters that are being mapped to are completely irrelevant, as long we have consistent functions to map and reverse map as required.

It is also useful, when working with these kinds of remappings, to give some thought to the specifics of the lexicographical sort order that is actually being used, as different programs may interpret this differently. Do the capital letters come before the lower-case letters? Do the numbers come before the letters? What about punctuation? The real issue here is beyond this text, except to say that different sort programs may handle this differently, and many different schemes are possible (Hopcraft et al., 2006). Perl's lexical sort is identical to the ASCII numeric sort order. However, other sort algorithms, such as UNIX sort, or those used in other languages such as Java, may handle this task differently. Mixing sort implementations without comparing and evaluating their exact properties is not recommended.

Table 2. Mapping of pitches to single-char code.

a	→d	d	→e	f#	→f
a#	→k	d#	→b	g	→i
b	→h	e	→a	g#	→j
c	→g	f	→m	r	→c
c#	→l				

5.2 Choice of feature(s)

We perform our searches using both pitch and duration intervals simultaneously. The use of both interval types greatly increases the accuracy of our system, and significantly filters out large numbers of false matches.

Matching on intervals has significant advantages over directly using pitches and note durations. Pitch intervals, which define the distance between pitches, is generally preferable, as interval searches are transposition-invariant. Similarly, rhythmic intervals, created by dividing each duration by the preceding value, will locate rhythmically augmented and diminished versions of melodies. Additionally, we also calculate the inversion, retrograde, and retrograde inversion of each phrase. Comparisons made possible using interval matching are given in Table 4.

While not necessary as yet in this project, other musical characteristics, such as dynamics or performance markings may also be mapped. A more comprehensive treatment of the topic of features for symbolic music analysis, including many other derived features, is

Table 3. A remapped musical sequence.

Pitches	c#	f#	a	c#	d	c#	b	a	g#	f#	a	c#	b	a	g#	f#	e#
Encodings	l	f	d	l	e	l	h	d	j	f	d	l	h	d	j	f	m

available in Conklin's work on multiple viewpoints (Conklin & Witten 1995).

5.3 Multidimensionality, large Cartesian sets and unicode

Most string-processing algorithms are designed to operate on a consecutive stream of tokens, such as a sequence of characters. While this method can work well for simple monophonic pieces, it has limited applicability to complex melodic structures. Even finding a representation that can handle two dimensions, such as pitch and duration intervals, without even considering other elements such as dynamics, articulations, or harmony can be difficult (Cambouropoulos et al., 2001).

One approach is to build the Cartesian product of multiple symbols (Xenakis, 1960; Triviño Rodriguez & Morales-Bueno, 2001). An example of this is demonstrated in Table 5. However, this can rapidly lead to large token sets that may exceed the limits of single-character representations.

For instance, consider a search set combining pitch and rhythm values. A set of five basic rhythms (whole, half, quarter, eighth, sixteenth), each with an augmentation dot, combined with a set of twelve pitches, gives a combined Cartesian set of 120 symbols. If specific octaves are taken into account the number would be much larger! There are 128 characters available in the ASCII set, including both upper and lower-case letters, numbers, and punctuation (although only approximately 90 characters will appear visually, which may be a factor in some circumstances). Our system requires approximately 450 pitch/duration combinations. While the basic ASCII set is enough for single-dimensional tasks, it is not sufficient for large sets of tokens.

One simple, and perhaps obvious solution to this problem is to use a more advanced text encoding format such as Unicode. Unicode is a multi-linugal text representation that standardizes the use of multiple characters per symbol, and has found wide acceptance on the web, with many standard parsing libraries available for most languages. The most common Unicode standard, UTF-8, encodes tens of thousands of characters, which should be enough for practically all music applications.

Table 4. Melodic search possibilities.

Pitch interval	transposition, inversion
Duration interval	retrograde, augmentation, diminution
Combined	retrograde inversion, other

Table 5. Token calculation.

Notes	e4 g2
Pitch interval	3 semitones
Duration interval	2.00000 doubling
Combined token	3_2.00000

6. Finding common phrases

6.1 Tries, suffix tries and suffix trees

A *trie*, taking its name from the word *retrieval* (Fredkin, 1960), is a rooted, multiway tree structure for storing and searching lexicographically-ordered strings. Tries are often used to quickly test for the existence of string sequences, such as are used by spell-check functions, or as indexes for online dictionaries.

Each node of the trie represents a symbol x of the string's alphabet Σ, and each path from the root to a leaf represents a single substring. A chain of nodes extending from the root to a branching node represents a prefix common to more than one sequence. In other words,

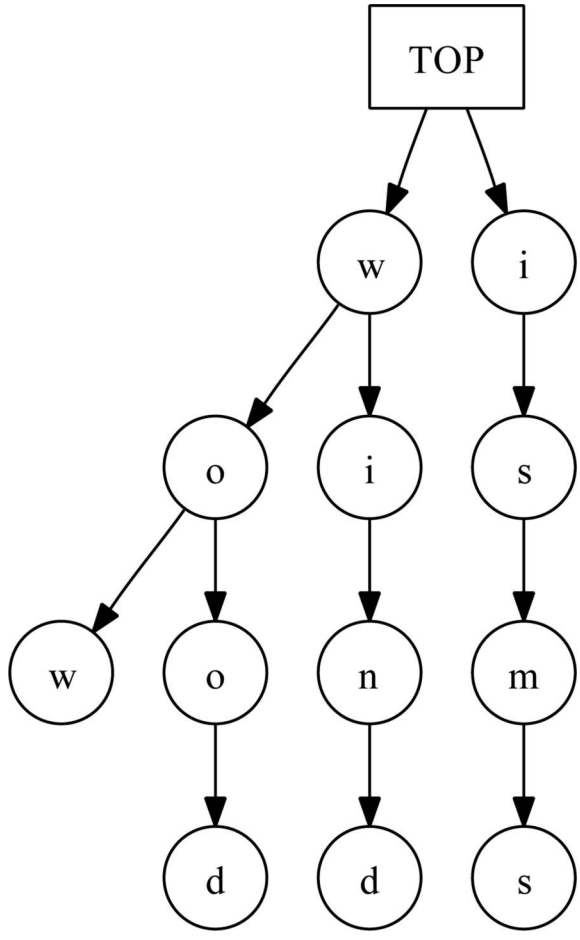

Fig. 2. An example trie for the sequences *wow*, *wood*, *wind*, and *isms*.

branches in the trie occur at the points where strings with different suffixes diverge.

An example trie for the strings *wow*, *wood*, *wind*, and *isms*, is shown in Figure 2. In this case the words *wow*, *wood* and *wind* share the same common prefix *w*, while only *wow* and *wood* share the prefix *wo*. Because every node with multiple children represents a common prefix between two strings, these prefixes can be reconstructed by concatenating all of the nodes between the root and the branching node.

A string $X = x_0x_1x_2 \ldots x_m$ over the alphabet Σ will have length $m = |x|$. The set of *suffixes* of X is a collection of substrings $x_0..x_m$, $x_1..x_m$, $\ldots$ x_m, formed from each successive symbol of the string to the end of the string. The symbol ϵ is used to represent the empty string, and is added to the end of each substring as a terminating symbol. Note that we start our indexes at 0, so that m keeps the same value as the original string length.

As an example, the set of suffixes for the string 'bananas', length $m = 7$, is shown in Table 6. In practice, the procedure for forming each new suffix is to repeatedly remove the first symbol from the string.

A *suffix trie* is created by inserting each of the above suffixes into a trie structure, such as that shown in Figure 3. This has the effect of grouping substrings together by their prefixes on the same branches. Each leaf of the trie then refers to a single suffix of the string, with the position stored in the leaf of the suffix. In order to know where each suffix came from, the position of each suffix within the original string is stored in each leaf as a index.

Suffix tries can be used to solve many different problems, including the longest common substring problem (Gusfield, 1997). A more compact, memory-efficient version of the suffix trie, known as a *suffix tree*, groups together chains of nodes that do not contain branches into single nodes. This leaves only nodes in the tree that have multiple children. There are several algorithms available for constructing such trees, including some that work in linear time (McCreight, 1976, Ukkonen 1992, Crochemore & Rytter, 2002). Suffix trees and tries have also been previously applied to music search problems, albeit on smaller collections (Lemström & Laine, 1998; Lemström, 2000; Conklin & Anagnostopoulou, 2001).

Table 6. Suffix substrings.

Indices	Substring
$x_0..x_7$	b a n a n a s ϵ
$x_1..x_7$	a n a n a s ϵ
$x_2..x_7$	n a n a s ϵ
$x_3..x_7$	a n a s ϵ
$x_4..x_7$	n a s ϵ
$x_5..x_7$	a s ϵ
$x_6..x_7$	s ϵ
x_7	ϵ

6.2 Suffix arrays

An alternative variant of the suffix tree or trie is the *suffix array* (Manber & Myers, 1991). This is a linear array containing all of the suffixes of a particular string that have been sorted lexicographically. This array can be stored as a set of references to locations in the original string, or, if memory is less of a concern, as an actual list of text strings. One advantage of the latter approach for this kind of work is that it makes exploratory evaluation of the represented music much simpler than in the tree-based versions. Table 7 demonstrates a basic example, using the substrings from Figure 3.

Due to the lexicographical ordering of strings, all suffixes beginning with common prefixes will then appear on contiguous rows of the array. Testing for the existence

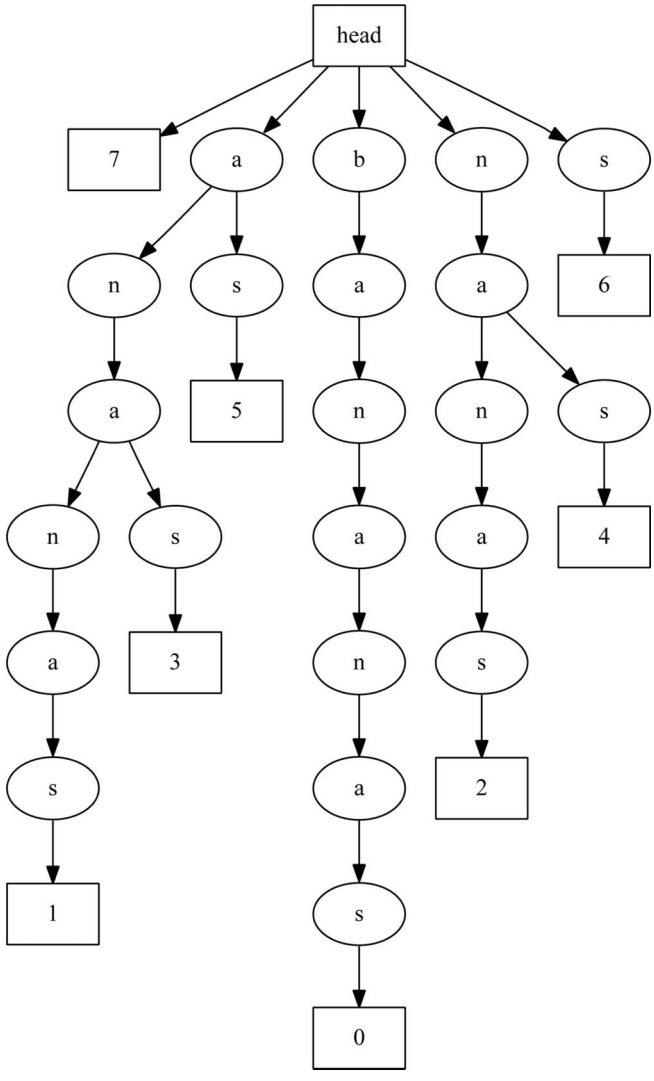

Fig. 3. A suffix trie for the word 'bananas'.

of particular strings can be solved very efficiently using a binary search algorithm. Many other string-matching problems can also be solved. For instance, the longest common substring can be easily found by comparing each pair of successive substrings and reporting the maximum-length pair.

There are several features of suffix arrays that make them attractive for our purposes. They can be easily extended to disk-based versions, an important consideration when working with extremely large data sets. Also, they can be easily made to work with existing highly-optimized string-processing programs, such as the UNIX sort program.

7. Suffix array insertion and metadata storage

To actually insert a mass into the array requires each stave to be cut into phrases. Phrase boundaries are preserved because we do not consider matches that overlap two phrases to be meaningful in this music, although this is not true in other types of music such as Jazz. Each phrase is then encoded using the mapping techniques and substring procedure described previously. In effect, this is equivalent to taking every stave of every mass and reconfiguring them as a single long string, where each note of this string gets a separate line in a (very long) array. In practice, it is not necessary to keep the entire suffix of every note, but only a reasonable window of each one (this is why we use phrases). In fact, each phrase must also be inserted four times, once each for normal order, inversion, retrograde, and retrograde inversion, to insure that all four combinations will be matched.

In addition to the suffix array system, a second text database is kept of every note in the entire set of masses, including the mass number, movement number, stave, and note index within the stave. This is useful in retrieving information from the final match list. Each note entry is given its own identification number, and these are used to relate the two databases. This is accomplished by attaching the index number to the end of each suffix array phrase, separated by a tilde.

Table 7. Suffix substrings in a suffix array.

Indices	Substring
$x_1..x_7$	a n a n a s ϵ
$x_3..x_7$	a n a s ϵ
$x_5..x_7$	a s ϵ
$x_0..x_7$	b a n a n a s ϵ
$x_2..x_7$	n a n a s ϵ
$x_4..x_7$	n a s ϵ
$x_6..x_7$	s ϵ
x_7	ϵ

An excerpt of a final encoded, sorted array, with metadata indices, is shown in Table 8. Each melody in this group shares a common prefix of the encoding dlda, which translates to the pitches a,c#,a,e. Note that these could have come from anywhere in the entire set of masses.

To retrieve actual musical results requires the process to be reversed. That is, each note must be looked up by its associated index in the metadata database and unmapped. The common tokens are then considered a 'match'.

8. Example matches

Figure 4 shows a set of matches in the Benedictus of Missa *Virtute magna*. The alto entry is matched up to the end of measure 37 by the bass entry (Match 114), and up to the end of measure 36 by the soprano entry in measure 41 (Match 1065). The initial soprano entry in measure 33 is matched until the middle of measure 37 by the second alto entry, commencing in measure 38 (Match 388). The initial duo formed by soprano and alto (Matches 114 and 388) is repeated an octave lower by bass and alto (the second occurrences of Matches 114 and 388).[2] The fact that these two melodic matches recur in the same temporal relationship to each other lets us spot them as forming a repeated module.

In Figure 5, found in the Sanctus of Missa *Pater noster*, the bracketed melody is a match of *itself*, in retrograde inversion. The axis of inversion is G#/A♭. This segment is, in fact, a palindrome with a twist. This example is particularly interesting, in that it prompts us to ask questions that might not otherwise have occurred to us: are there many such seemingly-coincidental structures? Are they sufficiently distinctive to suggest that they were devised deliberately?

Table 8. An excerpt of a final, encoded, sorted array.

dldaddadfps ~ 1969
dldaddampdamepmppmpill ~ 16970
dldaddpilemepfpememepfpemamu ~ 11449
dldaddppdi ~ 2539
dldadduiudampdppgmadd ~ 1783
dldadpdeliufadsglll ~ 6559
dldadp ~ 10700
dldafppmepmadddamamppfpppmppill ~ 16387
dldallllmpplmallmadpgmpppqdp ~ 7218
dldalsd ~ 7135
dldalsd ~ 9871

[2]Each of these duos of course also contains a match within itself, due to imitation at the fifth.

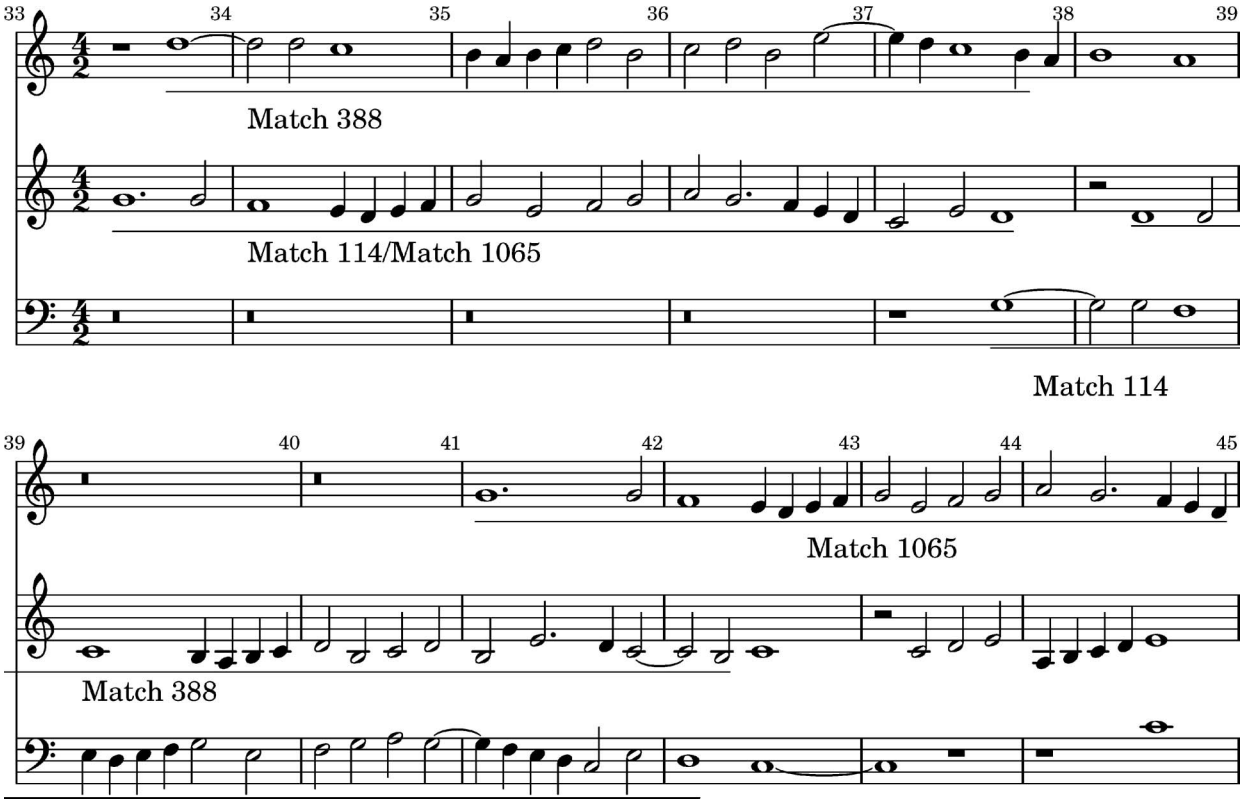

Fig. 4. Missa *Virtute magna*.

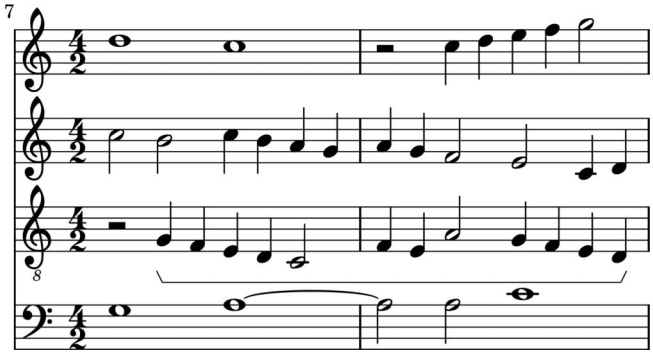

Fig. 5. Missa *Pater noster*.

Although our results are preliminary, some trends can already be spotted. An examination of long matches (over ten notes) showed that, as one would predict, the very longest matches are canons, such as those commonly found in the second Agnus Dei. From a knowledge of common shapes of points of imitation, one might expect the entries of all the voices in a point of imitation to yield quite long matches. However, unexpectedly, it appears that matches *within* a voice, i.e. repetition of a melody, tend to be longer than matches *between* voices. Measures 10–21 of the Kyrie of Missa *O Regem coeli* (Figure 6) will serve to illustrate this. The soprano melody which begins the example is in fact the second half of a melody which opens the whole movement, but this second half is used by itself as the basis for consecutive points of imitation. The bracketed segment is the longest repeated section of this melody—it occurs in the soprano. Meanwhile, the three-voice texture of the original presentation of this soprano melody (measures 10–14) is intensified to a four-voice texture, in which the original tenor entry (measure 11) is bolstered by the bass in parallel tenths (measure 19). In Palestrina's first book of motets, Peter Schubert observed such instances of 'heightening', in which a given melody forms the subject of two successive points of imitation, and the second of these shows an intensification, for example through an increased density of imitative entries (Schubert, 2007, pp. 530–531). The fact that the longest repeated segment is within the same voice underlines the repetition of the subject matter, against which the variation in the imitative structure becomes all the more apparent. Our preliminary data suggest that this technique is very commonly used when there are successive points of imitation on the same melody.

9. Match distributions and overall properties

It is interesting to note some of the overall properties of the matches found across the entire set of masses.

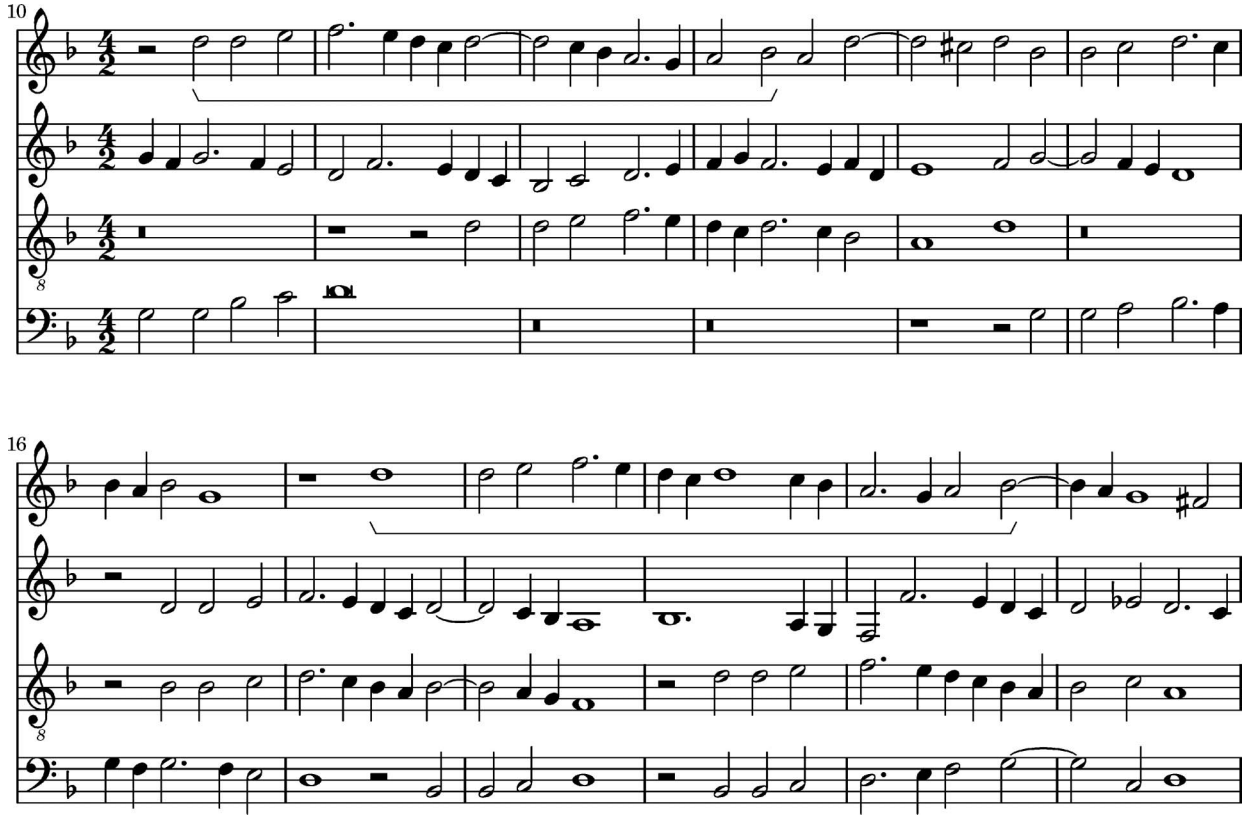

Fig. 6. Missa *O Regem coeli*, Kyrie, Measures 10–21.

Figure 7 shows the number of matches of each length that occur throughout the data set, down to a length of four notes. As one would expect, there is a dramatic rise in the number of matches found as the length of match decreases, and matches of length three or less are largely meaningless for our purposes. On the high end, there are a few matches of more than twelve notes. In general, matches of this length or longer tend to be imitated material within a Mass or Mass movement. The longest match found is of length 38, matching a reccuring Osanna between the Benedictus and the Sanctus of Missa *Beatus_Laurentius*.

Figure 8 demonstrates matches between masses with a minimum length of 6 (crosses indicate one or more matches). Note that many more matches occur within each mass group than between masses, indicated by the grouping along the central diagonal line. In general, match lengths are fairly evenly distributed across the entire set. Figures 9 and 10 show the same dramatic reduction of matches as the match length increases, which was indicated in Figure 7.

In Figure 11, the number of matches exceeding six notes in length is plotted against the Kyries of the Masses in a very rough chronological order (drawn from the publication volume of the Masses' first appearance). In the second graph, Figure 12, the number of matches exceeding six notes in length is plotted according to Mass genre

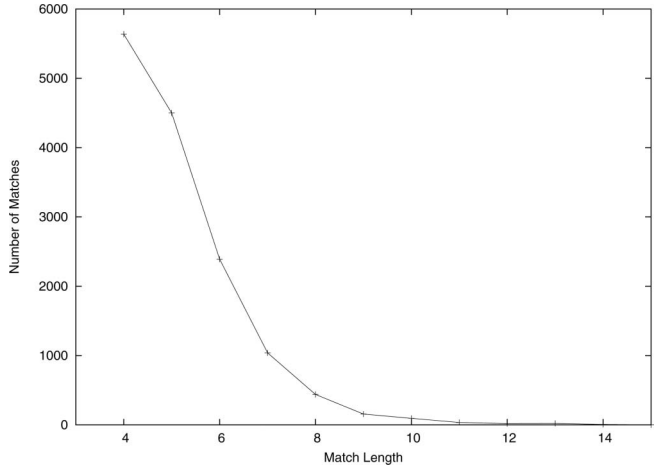

Fig. 7. Distribution of match lengths.

(Canonic, Free, Paraphrase, Parody, and Tenor). There does not appear to be a correlation between the number of exact matches, and the rough chronology of the Masses. A more precise investigation into the types of structures formed by these matches needs to be undertaken to reveal any patterns. The grouping of Masses by genre, however, shows that Tenor Masses (concentrated at the right end of the X-axis) appear to have fewer long matches than other genres. This confirms the intuitive assumption that a

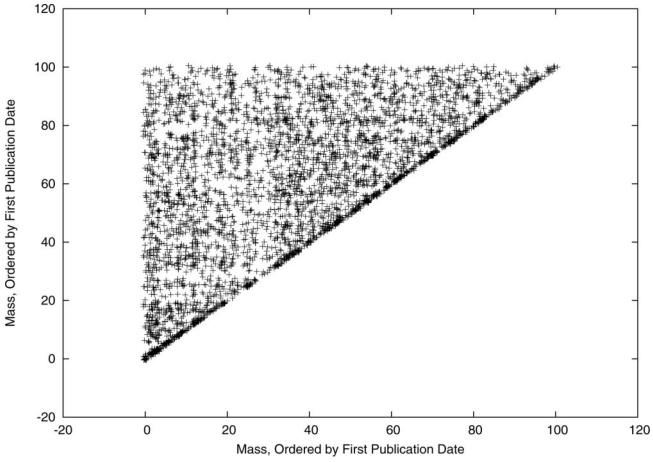

Fig. 8. Matches between masses: length ≥6.

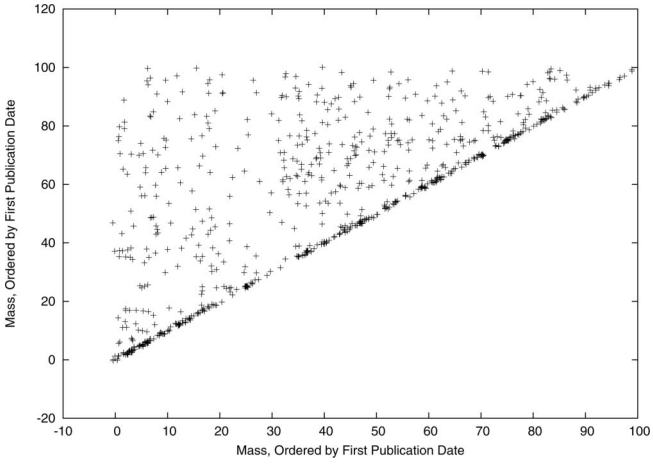

Fig. 9. Matches between masses: length ≥8.

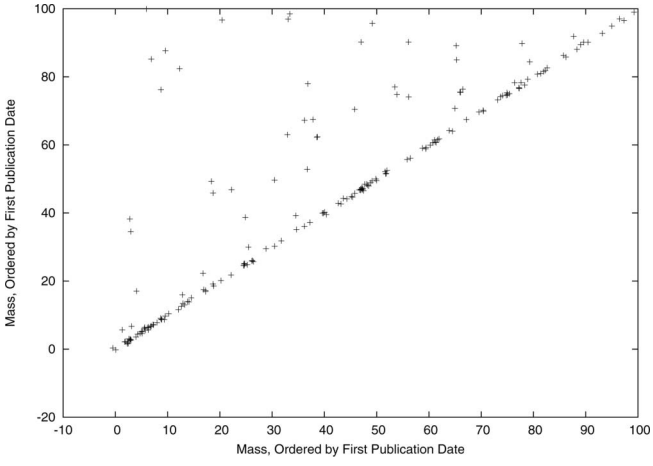

Fig. 10. Matches between masses: length ≥10.

cantus firmus makes imitative structures more difficult, thanks to the restrictions that a long-note tenor places on the counterpoint. Once modules are taken into consideration, we expect to find an even stronger tendency here.

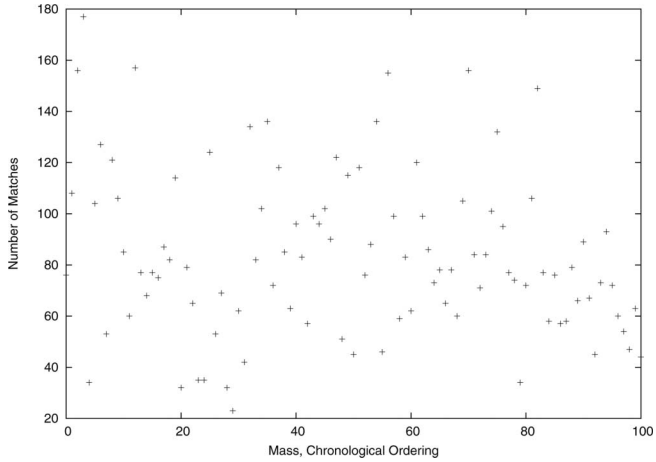

Fig. 11. Matches per mass, according to first publication date: length ≥6.

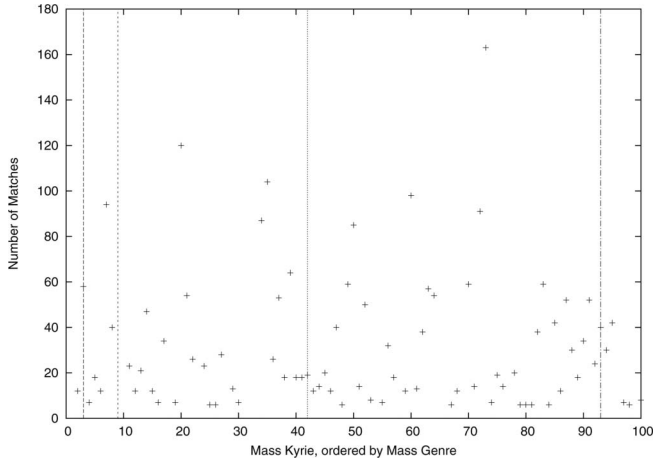

Fig. 12. Kyrie matches per mass, ordered by mass genre: length ≥6.

10. Future directions

Our eventual goal is to investigate structural properties of the masses in the context of contrapuntal *modules*. These are short vertical combinations of melodic segments, and are repeated subject to various transformations such as inversion, transposition, or ornamentation. These modules have been investigated recently in several important research projects, for example Owens (1997), Schubert (1994), Judd (1992, 1998), and Lessoil-Daelman (2002). Most recently published, and most relevant to this study, is work undertaken at McGill University by Peter Schubert (2007). Schubert cites several theorists (including Zarlino, Cerone, Montanos, and Santa Maria) to support the idea of looking for repeated chunks of counterpoint (Schubert, 2007, p. 485), and gives complete modular

analyses of Palestrina's 1564 first book of motets, demonstrating that 'repetition is the basic structural principle of this repertoire' (Schubert 2007, p. 486). Reliable identification of melodic repetition is the most important step along the way to identifying contrapuntal modules, since a module is in the end simply a vertical configuration of two or more melodic segments.

Ultimately, we would like to determine to what extent various patterns of repetition are characteristic specifically of Palestrina, and how his preferences developed throughout his career. These can then be compared to the works of contemporaneous composers, and also be linked to studies of earlier repertories. We anticipate that our approach to answering these structural questions across entire corpora of contrapuntal music will lead to a far more systematic consideration of concepts such as compositional practice, style, and style change than has previously been accomplished.

References

Cambouropoulos, E., Crawford, T., & Iliopoulos, C.S. (2001). Pattern processing in melodic sequences: Challenges, caveats and prospects. *Computers and the Humanities*, *35*(1), 9–21.

Conklin, D., & Anagnostopoulou, C. (2001). Representation and discovery of multiple viewpoint patterns. In *Proceedings of the International Computer Music Conference*, Havana, Cuba (pp. 479–485). San Francisco: International Computer Music Association.

Conklin, D., & Witten, I. (1995). Multiple viewpoint systems for music prediction. *Journal of New Music Research*, *24*(1), 51–73.

Crochemore, M., & Rytter, W. (2002). *Jewels of Stringology*. Singapore: World Scientific.

Fredkin, E. (1960). Trie memory. *Communications of the ACM*, *3*(9), 490–499.

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge: Cambridge University Press.

Hopcraft, J.E., Mowani, R., & Ullman, J.D. (2006). *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Boston, MA: Addison-Wesley Longman Publishing.

Judd, C.C. (1992). Josquin des Prez: Salve Regina. In M. Everist (Ed.), *Music Before 1600* (pp. 114–153). Oxford: Blackwell.

Judd, C.C. (1998). Musical commonplace books, writing theory, and 'silent listening': The polyphonic examples of Glarean's Dodecachordon. *Musical Quarterly*, *82*, 482–516.

Knopke, I. (2008). The perlhumdrum and perllilypond toolkits for symbolic music information retrieval. In *Proceedings of the International Symposium on Music Information Retrieval*, Philadelphia, USA, pp. 147–152.

Lemström, K. (2000). *String matching techniques for music retrieval* (PhD thesis). University of Helsinki, Finland.

Lemström, K., & Laine, P. (1998). Musical information retrieval using musical parameters. In *Proceedings of the International Computer Music Conference*, Ann Arbor, USA, pp. 341–348.

Lessoil-Daelman, M. (2002). *Une approche synoptique des motifs et des modules dans la messe parodique* (PhD thesis). McGill University, Canada.

Manber, U., & Myers, G. (1991). Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, *22*(5), 935–948.

McCreight, E.M. (1976). A space-economical suffix tree construction algorithm. *Journal of the ACM*, *23*(2), 262–272.

Owens, J.A. (1997). *Composers at Work*. Oxford: Oxford University Press.

Palestrina, G.P.d. (1938–1981). *Le opere complete di Giovanni Pierluigi da Palestrina* (33 Vols.). Rome: Fratelli Scalera.

Schubert, P.N. (1994). Mode and counterpoint. In C. Bernstein & D.W. Hatch (Eds.), *Music Theory and the Exploration of the Past*. Chicago: University of Chicago Press.

Schubert, P.N. (2007). Hidden forms in Palestrina's first book of four-voice motets. *JAMS*, *60*(3), 483–556.

Triviño Rodriguez, J.L., & Morales-Bueno, R. (2001). Using multiattribute prediction suffix graphs to predict and generate music. *Computer Music Journal*, *25*(3), 62–79.

Ukkonen, E. (1992). Constructing suffix trees on-line in linear time. In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture—Information Processing '92* (Vol. 1, pp. 484–492). Amsterdam: North-Holland.

Xenakis, I. (1960). Elements of stochastic music. *Grabesaner Blätter*, *18*, 84–105.

Copyright of Journal of New Music Research is the property of Routledge and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.