Looking for patterns in plainchant

Ruth M. Ripley, Department of Statistics,
University of Oxford
ruth@stats.ox.ac.uk

Emma C. Hornby, Christ Church,
University of Oxford

John Caldwell, Faculty of Music,
University of Oxford

Abstract

Musicologists undertaking detailed analyses of medieval manuscripts containing Western liturgical chant would welcome computer assistance to enable them to make swift comparisons of large amounts of unfamiliar material. The early manuscripts employ several different notations, using *neumes* rather than our modern notes. The earliest notations do not specify the exact pitch or rhythm of the music, only the approximate contour of the melody. Later notations are more precise regarding pitch but contain less information about the relative length of the notes. Most automatic analysis of musical sequences is based around pitch, rhythm and harmony: it is clear that a novel approach is required for this repertoire. We show that sequence-alignment algorithms can be employed to analyse this music, using a metric based just on the neumes. We can search for near-matched alignments of specific short phrases of music, or use clustering techniques to find groupings based on optimal alignments between pairs of chants or parts of chants. Computation is split between Visual Basic for the user interface, C for the algorithms and R for the statistical routines.

Keywords: cluster analysis, computer-assisted musical analysis, dynamic programming, sequence alignment.

1 Introduction

Musicologists undertaking detailed analyses of manuscripts of Western Christian liturgical chant dating back to the ninth century CE would welcome computer assistance with the task. In the early manuscripts a variety of different notations are used, which do not specify the exact pitch of the music, only the approximate contour of the melody. The sheer bulk of the material makes the idea of computer based tools attractive: ideally one would like to scan the manuscripts and identify both notational and musical variations and similarities automatically. The reality is rather different: an optical character recognition system is a large project in itself and even deciding how to encode the notation for automatic analysis is by no means straightforward. Since the notation provides only imprecise information regarding both pitch and rhythm we require a specially designed representation system for this music.

We have developed tools which can be used with this repertoire, which enable us to fi nd similarities between chants, or to examine their internal structure. The tools are based on sequence alignment algorithms, borrowed from computational biology. This type of algorithm has been used by others, including Mongeau and Sankoff (1990) and several listed by Cambouropoulos *et al.* (2001) in their Appendix, to analyse more modern music, with distance measures based on differences of pitch and rhythm. We use a measure based on the visual similarity or identity of the neumes, and show how the algorithms can be adjusted to solve various different matching problems. The algorithms are used to define a similarity measure between chants or parts of chants: from this a dissimilarity measure is calculated, for use in multivariate statistical methods. Hierarchical clustering techniques have been employed to construct trees showing relationships between and within highly structured chants. Partitioning about medoids has been used to group the phrases shared by chants in order to facilitate musicological analysis.

2 The Data

Ecclesiastical chants are settings of the Latin texts of the liturgy of the early Christian church. They are monophonic (i.e. only one line of music at a time), and do not follow modern Western tonality. Some of the chants are built from common melodic phrases: such chants are termed *formulaic*. The identification of these common melodic phrases offers one challenge to our analysis.

The chants are notated using signs called *neumes* which in the early stages simply represented one or more notes in a known pattern of relative direction of pitch. Information about the actual size of the intervals between successive notes within a neume, or about the direction of the melody between successive neumes was not given. Various other markings on the manuscript express nuances of performance such as the relative lengths of the notes. The actual shape of the signs varied from place to place, but all notations follow one of two basic systems: symbolic (where there is nothing inherent in the sign to define its meaning) or iconic (where the sign is a graphical depiction of the notes represented). There is still some debate as to which notations belong to each system (Treitler, 1982; Levy, 1987). It is possible that chants could have the same melody expressed by different neumes in different notations. Figure 1 shows part of a manuscript known as Saint Gall 359 (published in fascimile in Anon (1924)), which uses a very early type of (symbolic) notation. (Both manuscript and notation take their name from the location of the monastery where the manuscript is found.) This is the notation with which we began our investigation.

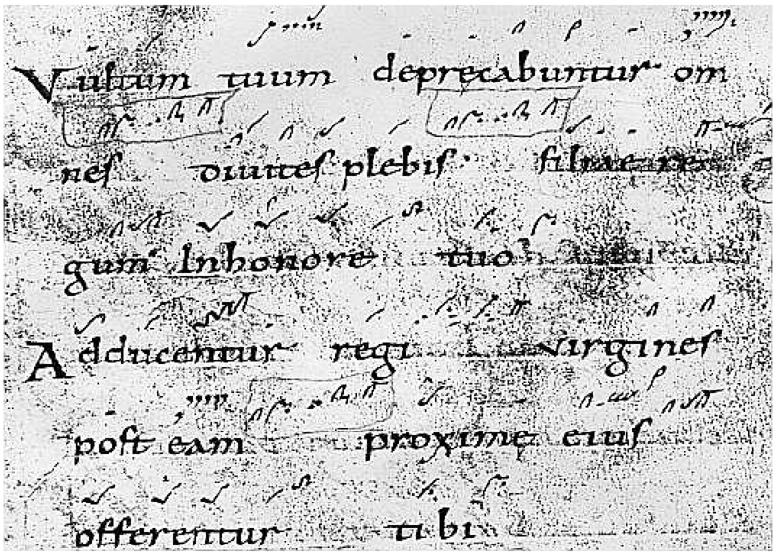

Figure 1: Part of the chant *Audi filia* (with some ‘interesting’ repeats marked)

Later manuscripts contain more information about the pitch of the notes, using the placement of the neumes to indicate the size of the intervals. However, the gain of more precise information about pitch is balanced by the loss of the more subtle details such as the length of the notes which were shown in the earlier notation by modifi cation of the signs or the addition of letters.

There are several families of chants, of which Gregorian is now the most widely known, as it was eventually adopted as the offi cial chant of the Western church. At the stage of the early manuscripts, we are interested in the differences between the families and manuscripts as well as the structure of the individual chants and groups of chants.

2.1 Approach

The aim of the overall work is to create a database of manuscripts, along with tools to analyse them. In the absence of software to enable us to scan the manuscripts directly into a database in a usable format, we have created a program to allow the entry of chants in great detail. The analysis is conducted on sets of sequences of integers, but interpretation of the results in musical notation is essential, which necessitates coding and storage of as much detail as possible. We have encoded several chants, enough to enable us to develop the analysis tools. A TrueType font has been created from hand-drawn neumes scanned into bitmaps, using High-Logic's Font Creator Program (High-Logic, 2002). which facilitates display and printing of our results.

At present the database is stored in a text format which is very easy to edit when the required format changes, as occurs frequently during this experimental phase of the work. It will be straightforward to convert this format to use Access databases or simpler comma-separated text files, either of which can be accessed via Microsoft ADO (ActiveX Data Objects) (Microsoft Corporation, 2003) database interfaces in the analysis programs.

2.2 Data format

It is simplest to think of the data as a hierarchical structure as shown in figure 2. A chant is made up of *verses* each of which are made up of *syllables* with each of

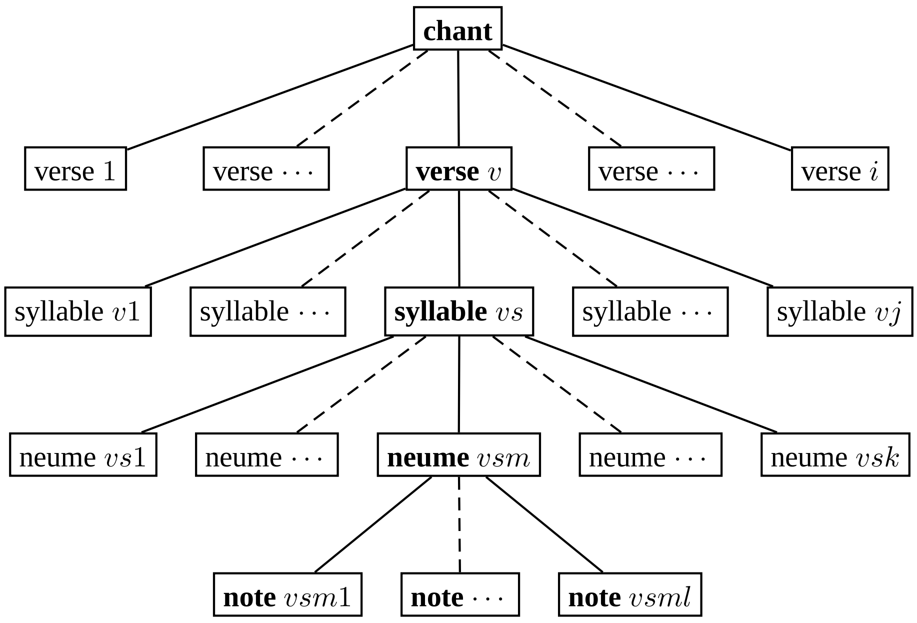

Figure 2: Structure of a chant: Note that, in order to save space in the diagram only one complete subtree is shown.

which are associated *neumes* each of which represents one or more *notes*. Some chants have another level of organisation, the *phrase*, which is between verse and syllable.

Appropriate attributes are stored at each level, for example, each syllable will have a corresponding syllable of text, and each note may either be long, short or normal length, and its pitch relative to the previous note may be up, down, equal or unknown. There are about 30 basic types of neumes, but each comes with many variants. Details of which variant has occurred are stored in the database but not used for matching purposes. Display or print routines do show which of the variants was used, along with the corresponding text. The basic neumes are shown in figure 3.

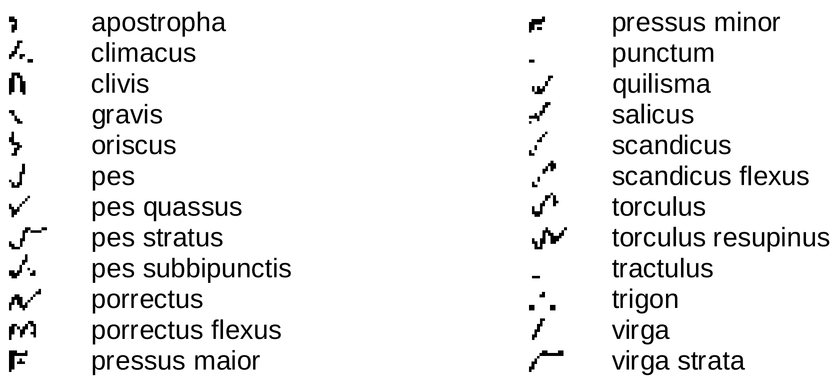

Figure 3: Different types of neumes

3 Analysis

Our aim is to find structure in the data based on near matches. Naive matching algorithms such as simply counting the number of neumes that matched in any of the possible alignments of the two sequences did not identify interesting relationships. We need to find the *best alignment* of the two sequences, allowing occasional insertions (or deletions, depending on your point of view) or mismatches, as shown in figure 4. Here there are seven insertions. We have not yet found mismatches preferable to two insertions, one in each sequence.

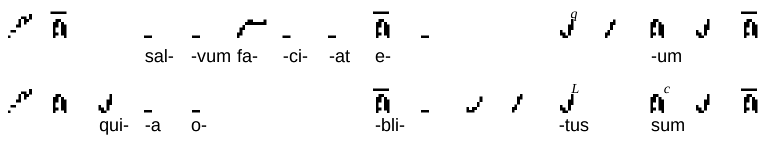

Figure 4: Alignment of two parts of chants, showing several insertions.

Once the problem was identified, the solution was clear: our problem was essentially the same as that of protein sequence alignment, and we investigated computational biology algorithms. We have used the *Needleman-Wunsch* and *Smith-Waterman* algorithms, which are both dynamic programming algorithms. We describe our use of the algorithms below: for further details the reader is referred to Waterman (1995).

3.1 Dynamic programming alignment algorithms

We consider our neumes (see table 3) as an alphabet, and for convenience add a null character, $-$, to represent a blank. We define a distance on our alphabet: for any two neumes, a and b, and null $-$

$$d(a,a) = 0$$

$$d(a,b) \geq 0$$

$$d(-,a) = d(a,-) > 0$$

Here a penalty for insertions is identified with the distance to '$-$', the null space-filler, which would be inserted in the other sequence opposite an insertion.

Then, we convert sequences $\mathbf{A}$ and $\mathbf{B}$ into $\mathbf{A}^*$, $\mathbf{B}^*$ by inserting null characters as required to create an alignment. $\mathbf{A}^*$ and $\mathbf{B}^*$ will have the same length, L. The distance between the sequences is then defined by

$$D(\mathbf{A}, \mathbf{B}) = \min \sum_{i=1}^{L} d(a_i^*, b_i^*)$$

where the minimum is taken over all possible alignments of $\mathbf{A}$ with $\mathbf{B}$, i.e. all possible $\mathbf{A}^*$, $\mathbf{B}^*$. Equivalently, we can define a *similarity* measure S by

$$\begin{aligned} s(a,a) &> 0 \\ s(a,b) &< 0 \text{ at least for some } a, b \\ s(-,a) &= s(a,-) < 0 \\ \text{and } S(\mathbf{A}, \mathbf{B}) &= \max \sum_{i=1}^{L} s(a_i^*, b_i^*) \end{aligned}$$

Here similarities compared with null place-fillers are negative.

The general algorithms depend on the way in which insertions are handled: in our case there has been no need to treat long stretches of insertions differently from individual ones, although this is usual in computational biology. For our purposes the penalty for insertion of k neumes is just the sum of the individual penalties, which may differ between neumes depending on the 'rarity' of the neume. This additivity of penalties simplifies the algorithms considerably.

We require two different algorithms: one which will align the whole of two sequences, a ‘global’ algorithm, and one to find the best matching subsequences of each, a ‘local’ algorithm. When looking for a global match the algorithms using distance are equivalent to those using similarity measures. In the local case, however, similarity is much easier to use, largely because it is clear that only positive scoring alignments need to be considered.

The problems we need to solve are:

1. find a motif in a chant: align a short sequence within a longer one, and find a score for each alignment. Identify all non-overlapping alignments of the complete short sequence which score more than some specified value.

2. find the score of an alignment between the whole of two chants, not penalising the alignment simply because the chants are of different lengths

3. find all non-overlapping alignments between any subsections of two chants which score more than some specified value

We want to use the scores found to derive a metric on the space of chants, which can be used in multivariate analysis procedures. We have used a metric derived from the global matches (2 above), and, for different types of chants, the maximum score (3, above).

The first two problems listed employ the global algorithm, the third uses the local algorithm. The differences between the first two are dealt with by alterations to the initialisation of the matrix used (explained below), and the method used to determine the similarity score from the matrix produced by the algorithm. We work with similarities throughout, and convert to dissimilarities later if required for the statistical analysis.

3.1.1 Global algorithm

The global algorithm, usually known as the *Needleman-Wunsch* algorithm, is:

Given two sequences $\mathbf{A} = a_1 a_2 \cdots a_n$ and $\mathbf{B} = b_1 b_2 \cdots b_m$, defi ne

$$\mathbf{S}_{i,j} = S(a_1 a_2 \cdots a_i, b_1 b_2 \cdots b_j).$$

Set

$$\mathbf{S}_{0,0} = 0, \quad \mathbf{S}_{0,j} = \sum_{k=1}^{j} s(-, b_j), \quad \mathbf{S}_{i,0} = \sum_{k=1}^{i} s(a_j, -)$$

Then set

$$\mathbf{S}_{i,j} = \max(\mathbf{S}_{i-1,j} + s(i,-), \mathbf{S}_{i-1,j-1} + s(i,j), \mathbf{S}_{i,j-1} + s(-,j)) \qquad (1)$$

The global similarity between two chants is given by $\mathbf{S}(n,m)$.

The version just described penalises overlapping stretches at both ends, which is not always desirable. In case 1 above, when we are looking for a short sequence in a long one we adjust the algorithm initialisation by setting (suppose the short motif is sequence $\mathbf{A}$)

$$\mathbf{S}_{0,j} = 0, \text{ for all } j.$$

This prevents penalties being imposed on the longer sequence before the match has begun. Once it has begun, we penalise insertions in either sequence until the fi nal neume of sequence $\mathbf{A}$ is reached. The score of such an alignment will be found not in $\mathbf{S}(n,m)$ but in the maximum value in $\mathbf{S}(n,j)$ over j. Once we have extracted this best alignment we can update the matrix $\mathbf{S}$ to fi nd the next best which has no matches in common with the first.

For case 2 above, we often know that interesting matches will be very similar, and will be likely to include the end of the sequences. We do not want to reduce the scores too much simply because one sequence is longer than the other. We initialise the matrix as in case 1 so we do not penalise insertions at the start of the longer sequence. We then use $\mathbf{S}(n,m)$ as the similarity: this does penalise insertions at the end of the longer sequence.

3.1.2 Local algorithm

The local alignment algorithm (the *Smith-Waterman* algorithm) is only slightly different, cutting off the sequences if 0 is reached. Here $\mathbf{S}(0,j) = \mathbf{S}(i,0) = 0$, and

$$\mathbf{S}_{i,j} = \max(0, \mathbf{S}_{i-1,j} + s(i,-), \mathbf{S}_{i-1,j-1} + s(i,j), \mathbf{S}_{i,j-1} + s(-,j)) \qquad (2)$$

Again we find the best aligned sections, adjust the matrix and repeat the process until some cut-off score is reached.

3.1.3 Unravelling the alignment

For some purposes we simply need the similarity score, while for others we want to display the two sequences in optimal alignment. To do this, we keep a second matrix $\mathbf{T}$ the same size as the similarity matrix $\mathbf{S}$, which records which of the three or four possibilities achieved the maximum in equation (1) or (2. The terms in the maximand in equation (1) are labelled by $k = 0, 1, 2$. The value of $\mathbf{T}$ is given by

$\sum 2^k$ over those k that achieve the maximum. In the local algorithm, $\mathbf{T}$ is set to 0 if $\mathbf{S}$ is 0. Thus $\mathbf{T}$ uses the numbers 0 to 7 to reference the different subsets of $\{0, 1, 2\}$. At the end we retrace a possible path through the matrix and 'unravel' an optimal alignment. If we have to choose between two or more possible directions while unravelling (more than one term achieved the maximand), it is possible to store the alternative possibilities and to follow them later. However, this leads to a vast number of alignments which differ only very slightly, and we have found it more useful to simply unravel one possibility.

3.1.4 Other distinct alignments

When looking for all occurrences of a motif in a chant, or seeking best matching subsequences we will want to examine other alignments which do *not* overlap with the first. This is done by recalculating the $\mathbf{S}$ and $\mathbf{T}$ matrices after unravelling the first alignment. It is not necessary to recalculate the whole matrices: we start with the beginning of the first alignment, and recalculate the lower right hand part from here. As we go, the term in (1) which corresponds to a match/mismatch is omitted for any matches we have already used: we store these as we retrace our path through the alignment. In each row we can stop recalculating once an entry is unchanged from the previous matrix, provided we have gone at least as far as we did on any previous row. Figure 5 may make this process easier to understand: further details of the 'unravelling' of the alignment and removal of overlapping repeats can be found in Waterman (1995).

	–	ʃ	/..	–	ꜰ:	–	/	J	ɔ	ɔ	ɔ	ɔ	ɔ	–	–	∩	/
–	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ʃ	-3	3	0	-1	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
/..	-6	0	6	5	2	1	0	-2	-3	-4	-5	-6	-6	-6	-6	-6	-6
–	-7	-1	5	7	4	3	2	0	-1	-2	-3	-4	-5	-5	-5	-7	-7
ꜰ:	-10	-4	2	4	10	9	8	6	5	4	3	2	1	0	-1	-10	-10

	–	ʃ	/..	–	ꜰ:	–	/	J	ɔ	ɔ	ɔ	ɔ	ɔ	–	–	∩	/
–	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ʃ	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
/..	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6
–	-7	-7	-7	-7	-5	-6	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7
ꜰ:	-10	-10	-10	-10	-10	-8	-9	-10	-10	-10	-10	-10	-10	-8	-8	-10	-10

Figure 5: Part of the matrix $\mathbf{S}$, before and after recalculation.

3.1.5 Parameters

The matching algorithms use parameters defining the similarity between two neumes: these values are tuned to make useful matches more likely. We have tried two systems: the first is a common score for each match and penalty for insertion or mismatch, apart from one or two neumes which we treated specially. The *apostropha* usually occurs in multiple groups and tends to overwhelm the matches and was therefore downweighted a lot. Similarly the punctum, tractulus and virga were downweighted slightly.

We then followed this idea of downweighting further and used scores based on the number of notes represented by the neume: this proved more successful than the first scheme. We have not yet explored penalising partial matches less than total mismatches (some neumes can be considered as sub-neumes of others, e.g. *clivis* compared with *scandicus flexus*), but this could be done.

3.2 Using the similarities

The statistical procedures we have investigated require a *dissimilarity* measure rather than a similarity: we calculate this from the similarity matrix $\mathbf{S}$ by

$$\mathbf{D} = \sqrt{1 - \mathbf{TST}}, \text{ where } t_{ii} = \frac{1}{\sqrt{s_{ii}}}, t_{ij} = 0, i \neq j$$

We have two basic measures, one a global score between (most) of two sequences and one the score of a maximal scoring pair of subsequences. These two are very different in nature: in the former case we know that if sequence A is 'near' sequence B which is 'near' sequence C then A is 'near' C also. In the second case this does not hold: sequence A and C could have no non-trivial matching subsequences.

Formulaic chants

The first measure allows successful analysis of those chants built from common melodic phrases (*formulaic* chants). The distinctive part of these phrases often occurs at the end, so an end-weighted measure is appropriate. A dissimilarity matrix is created for the individual phrases in the chant or group of chants and hierarchical clustering is used to create a tree structure. This structure can be illustrated in a dendrogram and the dissimilarity between two sequences read off from the height on the dendrogram at which they join to form a group. There are several different methods which may be used to create hierarchical clusters: for details see Gordon (1981). The most successful for our purposes is *single linkage* clustering, where the distance between two clusters is defined as the shortest distance between a member of one cluster and a member of the other. Clusters are joined in increasing order of this distance. This part of the analysis was done using the `R` (Ihaka and Gentleman, 1996) function `hclust`.

Non-formulaic chants

Other chants are not built up from common phrases, and may not have a well-defined phrase structure at all. Interest focuses on sections of chant which are repeated, either internally or in more than one chant. With chants of this type we first search for all repeating sections which score more than a predefined limit, using the local search algorithm. We may search using the chant undivided, or split into verses. It is straightforward to exclude the complete matches of a chant or verse with itself, but all other repeats will occur twice, and are removed at a later stage. We define the similarity between chants or verses as the score of their maximal scoring subsequence. The similarity matrix of the set of chants or verses and the sets of similar subsequences can be displayed, but attempts to use the associated dissimilarity to form clusters are unsuccessful: this measure is not well approximated by an ultrametric. A different approach is therefore adopted: all the common subsequences are collected together, the duplicates removed (using `R`!), and then similarities are calculated for each pair of subsequences, using the global algorithm, with no penalties for overlaps either end on the longer subsequence. The resulting similarities (after conversion to dissimilarities) are then clustered by *partitioning about medoids* (using the `R` function `pam`). This searches for a set of representative objects or medoids among the observations, and forms clusters by assigning each observation to the nearest medoid. The goal is to find those representative objects which minimize the sum of the dissimilarities of the observations to their closest representative object. Full details of the algorithm may be found in Kaufman and Rousseeuw (1990, Chapter 2). In this way a representative member of each cluster of subsequences is selected for the user to review.

3.3 Division of analysis between systems

The user interface part of the analysis program is written in Visual Basic for ease of development and access to the database. Facilities are provided for the user to select the analyses required, specify the parameters for the algorithms and display and print the output. We began our experiments on this data using sequence alignment algorithms coded in S-PLUS, but found this very slow even on our small initial database. Changing nested loops to a diagonal vectorised approach improved things (details are given in (Venables and Ripley, 2000, pp.176–7)) as did the use of C. Currently we use C called from Visual Basic. The statistical procedures are performed using R (Ihaka and Gentleman, 1996) accessed from Visual Basic *via* a DCOM (Distributed Component Object Model) interface provided by Thomas Baier (Baier, 2001). This allows arrays to be passed back and forth between the Visual Basic program and R. Graphs plotted from R are displayed using the R graphics device and can be printed or saved. As our application provides the user interface we do not need a menu driven statistical package, and R has the great advantage of being available free. This division of labour exploits the strengths of each system.

4 Results

Formulaic chants

We used a collection of *tracts* as an example of formulaic chants. We illustrate the structure of one member of the family here: the same phrases occur in each of this set of tracts but graphical display is not useful with some two hundred phrases! Figure 6 shows a dendrogram for relationships between the phrases of the tract *Audi filia*, with the clusters outlined. The phrases are numbered by verse and then phrase within verse. The distances are based on the global algorithm, with initialisation as in the second task enumerated above. The clusters and sub-clusters identifi ed by this automated method have been found to be, in our limited experiments, very similar indeed to those determined by the musicologist. Occasionally the algorithm highlights similarities between different parts of the phrases from those expected by the expert. Such ignorance of prior information can be a useful feature of this type of automation. Figure 7 shows the beginning of a listing of the results from the tree diagram.

Non-formulaic chants

We used chants known as *alleluias* for this part of the work. These chants begin with a setting of the word *alleluia*; this is followed by a wordless decorated section of music called the *jubilus*. The remainder of the chant consists of one or two verses. Some of the verses contain material from the *alleluia* and *jubilus*, while repeating motives occur throughout. For our analysis the chants were split into

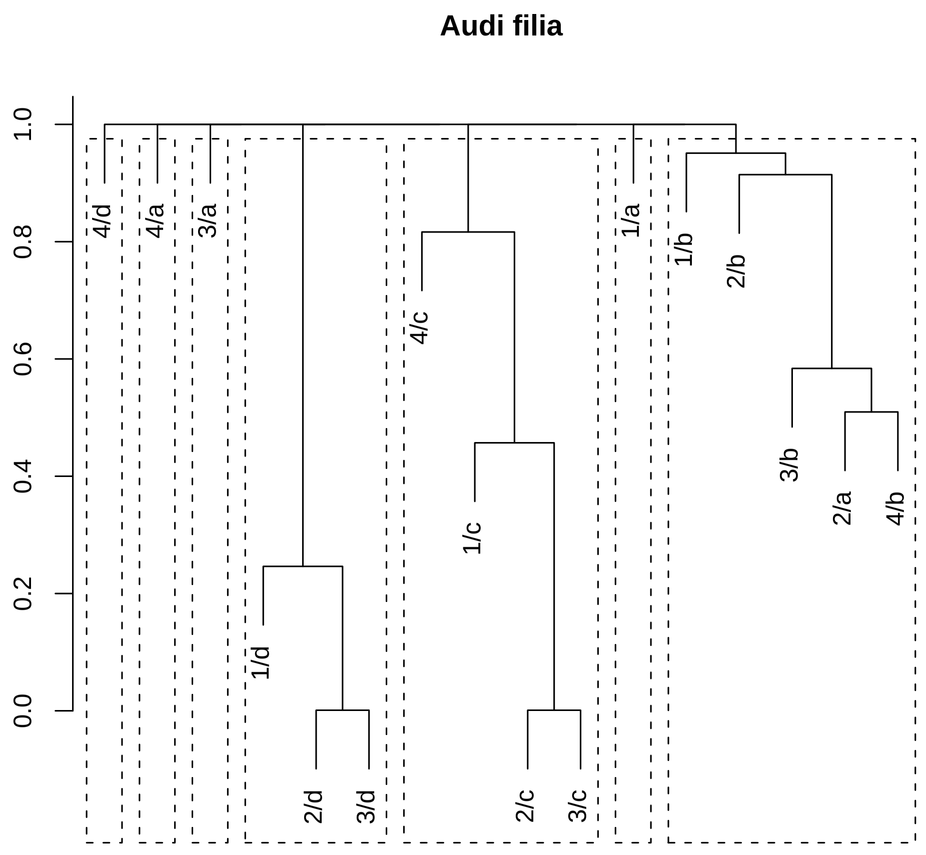

Figure 6: Dendrogram of phrases of *Audi filia*

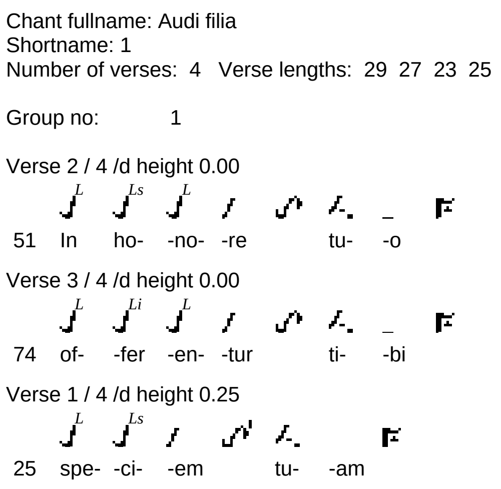
Chant fullname: Audi filia
Shortname: 1
Number of verses: 4 Verse lengths: 29 27 23 25
Group no: 1
Verse 2 / 4 /d height 0.00
L Ls L
51 In ho- -no- -re tu- -o
Verse 3 / 4 /d height 0.00
L Li L
74 of- -fer -en- -tur ti- -bi
Verse 1 / 4 /d height 0.25
L Ls
25 spe- -ci- -em tu- -am

Figure 7: Structure of *Audi filia*

verses with the alleluia and jubilus treated as the first verse. Highest scoring subsequences between each pair of verses were found, and the resulting subsequences clustered. Figure 8 shows the results of this analysis, applied to two of the alleluias which contain several repeating subsequences.

5 Discussion

The use of sequence matching algorithms along with clustering techniques has resulted in useful analyses of this chant material. When more music has been encoded the methods can be extended to include such techniques as multidimensional scaling.

The system is easily extensible to other chant notations: we have encoded some versions of the tracts in other notations in order to determine how to use our meth-

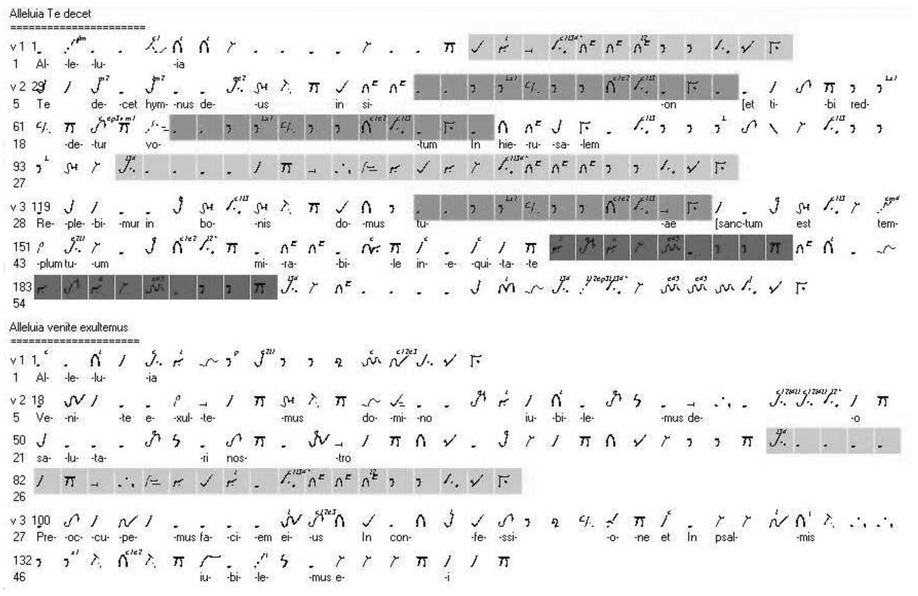

Figure 8: Three repeating subsequences (highlighted in different colours) of *Alleluia te decet* and *Alleluia venite exultemus*

ods in comparisons between differently notated chants. Two features we intend to incorporate are, firstly, comparison between chants with pitch information and those without and secondly, the consistency or otherwise of a chant with incomplete pitch information and a pitched version of the same chant.

In some cases we may be able to compare two very different notations by way of an intermediate one. Suppose, for example, we have three versions of a formulaic chant such as a tract, A, B and C, which use three different notations. Suppose further that the notation used in A groups notes into neumes but does not indicate exact pitch, that used in B shows exact pitch but indicates every note as a separate dot and that used in C includes both neumes and exact pitch. We would then find the distances from A to C and from B to C. The direct distance from A to B can then be left as infinite: single-linkage clustering should still be successful.

Acknowledgements

Ruth Ripley's work was partially supported by the Faculty of Music of the University of Oxford. Emma Hornby's work was partially supported by the British Academy. Professor Brian Ripley has provided much technical help.

References

Anon (1924) *Paléographie Musicale*, volume 2 of *series II*. Solesmes.

Baier, T. (2001) R (D)Com Interface (Version 0.99). Available at `http://cran.r-project.org/contrib/extra/dcom`.

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

Gordon, A. D. (1981) *Classification: Methods for the Exploratory Analysis of Multivariate Data*. London: Chapman & Hall.

High-Logic (2002) Font Creator Program (Version 3.1). Available at `http://www.high-logic.com/`.

Ihaka, R. and Gentleman, R. (1996) R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, **5**(3), 299–314.

Kaufman, L. and Rousseeuw, P. J. (1990) *Finding Groups in Data. An Introduction to Cluster Analysis*. New York: John Wiley and Sons.

Levy, K. (1987) On the origin of neumes. *Early Music History*, **7**, 59–90.

Microsoft Corporation (2003) Active X Data Objects 2.8. Available at `http://www.microsoft.com/data/ado/`.

Mongeau, M. and Sankoff, D. (1990) Comparison of musical sequences. *Computers and the Humanities*, **24**, 161–175.

Treitler, L. (1982) The early history of music writing in the west. *Journal of the American Musicologocial Society*, **35**, 237–279.

Venables, W. N. and Ripley, B. D. (2000) *S Programming*. New York: Springer.

Waterman, M. S. (1995) *Introduction to Computational Biology. Maps, Sequences and Genomes*. Boca Raton: Chapman & Hall / CRC Press.