Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music*

David Meredith
Department of Computing, City University, London.
dave@titanmusic.com

Kjell Lemström
Department of Computer Science, University of Helsinki.
klemstro@cs.helsinki.fi

Geraint A. Wiggins
Department of Computing, City University, London.
geraint@soi.city.ac.uk

March 10, 2003

Abstract

In this paper we give an overview of four algorithms that we have developed for pattern matching, pattern discovery and data compression in multidimensional datasets. We show that these algorithms can fruitfully be used for processing musical data. In particular, we show that our algorithms can discover instances of perceptually significant musical repetition that cannot be found using previous approaches. We also describe results that suggest the possibility of using our data-compression algorithm for modelling expert motivic-thematic music analysis.

1 Introduction

In this paper we give an overview of some algorithms that we have developed for discovering perceptually significant repeated patterns in polyphonic music. These algorithms can straightforwardly be adapted for music information retrieval (pattern matching) and data-compression. We'll begin in the next section by presenting some examples of perceptually significant repeated patterns in music which illustrate just how diverse this class of phenomena is.

Many music psychologists and music analysts (Bent and Drabkin, 1987; Lerdahl and Jackendoff, 1983; Nattiez, 1975; Ruwet, 1972; Schenker, 1954; Temperley, 2001) have noted that being able to identify the perceptually significant repetitions in a passage of music is often extremely important for achieving a rich understanding of it. However, the vast majority of exact repetitions in a piece of tonal music do, in fact, go unnoticed by listeners (see below).

It seems that most previous approaches to repetition discovery in music have been based on the assumption that the music to be processed is represented in the form of a string or a set of strings. We will briefly review some of these string-based approaches and demonstrate that there seem to be certain important types of musical repetition that are very difficult to find using such approaches. In particular, it seems that if one wants to find a wide range of different types of musical repeated pattern using a string-based approach, one generally has to run a variety of different algorithms on a number of different representations of the music.

In our work we have avoided such difficulties by adopting a *geometric* approach in which the music to be analysed is represented as a multidimensional dataset—that is, a set of points in a Euclidean space. We have found that by doing this we are able to, first, process polyphonic music as easily and efficiently as monophonic music; second, compute some of the repetitions that are difficult to find using a string-based approach; and third, essentially dispense with multiple representations because we can run the same small set of algorithms on various orthogonal projections of a single, rich multidimensional representation of the music.

In this paper we briefly describe two repetition discovery algorithms, `SIA` and `SIATEC`, that are based on this new approach. `SIA` computes all the maximal repeated patterns in a dataset and `SIATEC` computes *all the occurrences of* all the maximal repeated patterns in a dataset. We then briefly discuss what happens when you run these algorithms on music data. For detailed descriptions of the algorithms and precise definitions of the functions that they compute, see Meredith *et al.* (2002a) and Meredith *et al.* (2002b).

Our experiments suggest that the repeated patterns that we are interested in finding are often either equal to the maximal repeated patterns computed by `SIA` or derivable from them. However, typically, `SIA` also generates many patterns that are *not* musically interesting. Some post-processing is therefore usually required to isolate the interesting repetitions in the output of `SIA` and `SIATEC` and we suggest some heuristics that seem to be useful for doing this.

When these heuristics are incorporated into a data-compression algorithm based on `SIATEC` (which we call `COSIATEC`), we find that we can generate some quite interesting motivic and thematic music analyses. We then describe a pattern-matching algorithm based on `SIA`, which we call `SIAMESE`. This algorithm finds complete and partial matches of multidimensional query patterns in multidimensional datasets. Although in

*The algorithms described in this paper are the subject of an international patent application (Meredith *et al.*, 2002b).

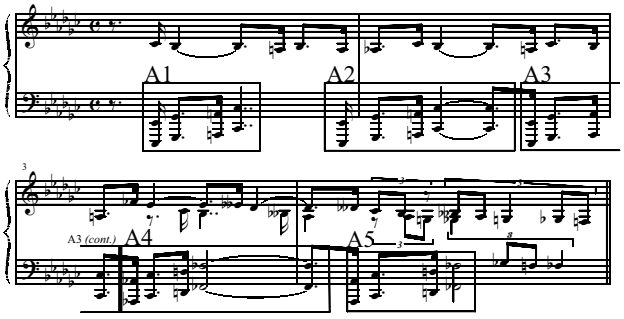

Figure 1: Opening bars of Barber's *Sonata for Piano*, Op.26

this paper we focus on the musical applications of these algorithms, it is important to remember that the algorithms are, in fact, quite general and can be used to process any data that can appropriately be represented in the form of a multidimensional dataset. We end the paper with some suggestions for further research

2 The diversity of musical repetition

Many music psychologists and music analysts (Bent and Drabkin, 1987; Lerdahl and Jackendoff, 1983; Nattiez, 1975; Ruwet, 1972; Schenker, 1954; Temperley, 2001) have stressed that identifying the significant repetitions in a piece of music is an essential part of achieving a rich and satisfying interpretation of it. Our work was originally motivated by the desire to develop a computational model of expert music cognition and it seems clear that one component of such a model would have to be able to discover perceptually significant repetitions, or instances of *parallelism*, as it is generally called by music psychologists and analysts.

However, the class of perceptually significant repetitions is a very diverse set. There are at least two reasons for this: first, the patterns involved in such repetitions vary widely in their structural characteristics; and second, there are many different ways of transforming a musical pattern to give another pattern that is perceived to be a version of it. For example, musical themes and motives can be truncated, augmented, diminished, inverted, reversed, embellished and so on.

The patterns involved in perceptually significant repetitions vary widely in size from small motives (such as the rising bass line in Figure 1) to whole sections of works containing hundreds of notes such as the exposition of a sonata-form movement. In polyphonic music in which the voices are unambiguously identifiable, the notes in a repeated pattern may all come from one voice or they may come from two or more voices. For example, in the *stretto* passage shown in Figure 2, each statement of the subject only contains notes from a single voice.

On the other hand, in Figure 3, each occurrence of

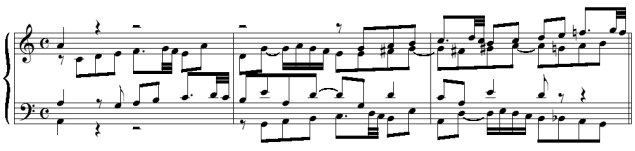

Figure 2: Bars 14–16 from Fugue in C major from Book 1 of J. S. Bach's *Das Wohltemperirte Klavier* (BWV 846).

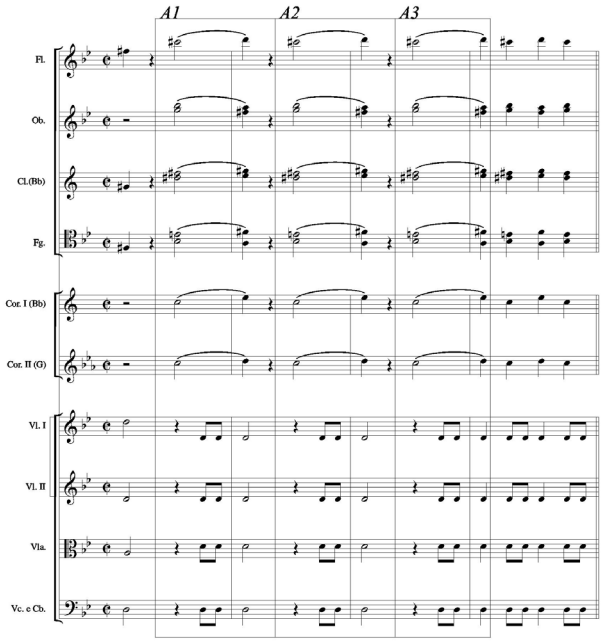

Figure 3: Bars 16–19 of the first movement from Mozart's Symphony in G minor, K.550.

the repeated pattern involves the whole orchestra and contains notes from 13 voices. These two examples also show that the occurrences of a pattern may overlap as they do in Figure 2 or they may occur consecutively, as they do in Figure 3. The occurrences of a perceptually significant musical repeated pattern may also be widely separated in the music as they are, for example, in the case of the exposition and recapitulation of a sonata-form movement.

Repeated musical patterns also exhibit different types of *compactness*. For example, the repeated pattern in Figure 3 is *temporally compact*—that is, it contains all the notes that occur within the time period spanned by the pattern. On the other hand, the rising bass pattern in Figure 1 is *bounding box compact*—it contains all the notes that occur within the pattern's bounding box when the music is represented as a graph of chromatic pitch against time. In Figure 2, the repeated overlapping occurrences of the subject exhibit yet another type of compactness: each occurrence contains all the notes that occur in a single voice within the time period spanned by the pattern. In Figure 5 we see themes being transformed by diminution and inversion and in Figure 4 we see an example of a simple melodic idea being embellished by *diminution* (Forte

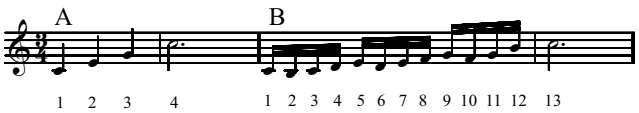

Figure 4: Example of diminution.

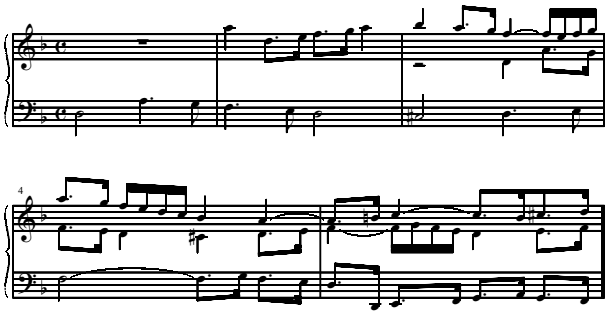

Figure 5: Bars 1–5 of Contrapunctus VI from J. S. Bach's *Die Kunst der Fuge* (BWV 1080).

and Gilbert, 1982), a process in which each long note is replaced by a sequence of shorter notes.

3 Most repetitions in music are not interesting

Although identifying the important repetitions in a piece significantly enriches a listener's understanding, not all repeated musical patterns are interesting and significant. For example, in Figure 6 the pattern consisting of the notes in elliptical boxes is repeated 7 crotchets later, transposed up a minor ninth to give the pattern consisting of the notes in square boxes. Clearly, this repetition is just an artefact that results from the other musically significant repetitions that are occurring in this passage such as, for example, the exact repetition of bar 3 in bar 4.

In fact, it turns out that, typically, the vast majority of exact repetitions that occur within a piece of music

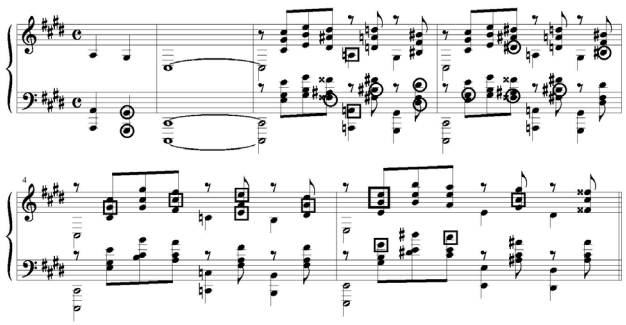

Figure 6: Bars 1–6 of Rachmaninoff's Prelude in C♯ minor, Op.3, No.2. The pattern consisting of the notes in square boxes is an exact transposed repetition of the pattern consisting of the notes in elliptical boxes.

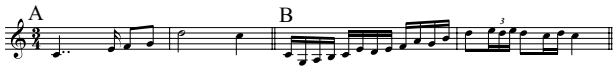

Figure 7: Two versions of a melodic idea separated by a high edit distance.

are *not* musically interesting. One of the motivations behind our work has been to develop algorithms that extract only the interesting repeated patterns of a particular type from the music. This involves formally characterising what it is about the interesting repetitions that distinguishes them from the many exact repetitions that the expert listener and analyst do not recognize as being important.

4 Previous approaches to repetition discovery in music

It seems that most previous attempts to develop a repetition discovery algorithm for music have been based on the assumption that the music to be analysed is represented as a string of symbols or as a set of such strings. An example of such an approach is Pierre-Yves Rolland's FlExPat program (Rolland, 1999). This program can find approximate repetitions within a monophonic source. It can also be used to find repeated monophonic patterns in a polyphonic work in which each voice is represented as a string. Also, it is capable of finding repeated monophonic patterns which contain 'gaps'. However it suffers from a few weaknesses. First, it cannot deal with unvoiced polyphonic music such as piano music. Second, it can only find patterns whose sizes lie within a user-specified range and if the range is set so that it allows patterns of any size, the overall worst-case running time goes up to at least $O(n^4)$. Third, like most string-based approaches to approximate pattern matching, it uses the edit-distance approach to compute the similarity between patterns. Unfortunately, such an approach is not typically capable of finding a match between a pattern and a highly-embellished variation on the pattern such as the one shown in Figure 7. In this example the edit distance between the two occurrences is rather large owing to the high number of insertions required to transform the plain version (A) into the embellished one (B). A program like Rolland's regards two patterns as being similar if the edit distance between them is less than some threshold k. However, for these two patterns to be considered 'similar' by Rolland's algorithm, this value of k would have to be set to at least 14 to allow for all those extra notes to be inserted. Unfortunately, this value would in general be too high because the program would then start regarding highly dissimilar patterns as being similar.

Hsu *et al.* (1998) describe a repetition discovery algorithm for music that uses the dynamic programming technique. This algorithm suffers from a number of se-

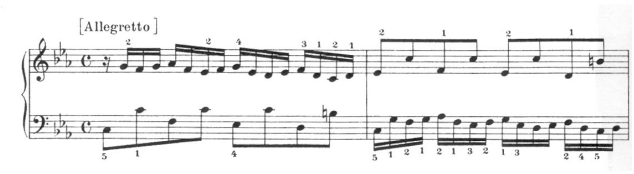

Figure 8: Bars 1–2 of the Prelude in C minor from Book 2 of J. S. Bach's *Das Wohltemperirte Klavier* (BWV 871).

rious weaknesses. First, it cannot be used for analysing unvoiced polyphonic music. Second, it is not capable, as described, of finding transposed repetitions. Third, it has a worst-case running time of $O(n^4)$ which means it is too slow to be used for analysing large pieces. Finally, it only finds repeated *factors* and therefore cannot find patterns 'with gaps'. That is, it can only find the repetitions of a pattern if the pattern contains all the notes in the piece that occur during the time interval spanned by the pattern.

Cambouropoulos's (1998) *General Computational Theory of Musical Structure* also contains a pattern discovery component, that, in the most recent incarnation of the theory, is based on Crochemore's (1981) 'set partitioning' algorithm. In Cambouropoulos's theory, this pattern-discovery algorithm is used to help with determining the boundaries of the segments that are then categorised. Crochemore's algorithm is very fast—it runs in $O(n \log_2 n)$ time. However, the algorithm does suffer from a couple of short-comings: first, it cannot find patterns with gaps; and, second, it cannot be used for finding patterns in unvoiced polyphonic music.

In a recent approach described by Conklin and Anagnostopoulou (2001), a number of different string representations each representing a different 'viewpoint' on the music, are derived from a rich representation of the music to be analysed. They then discover repeated factors in these various string representations and isolate those factors that occur most frequently. Their approach is interesting because it offers a crude way of identifying repeated patterns at higher structural levels than the musical surface—one of their viewpoints, for example, represents just the first note in each crotchet beat. However, such an approach would not be capable of finding the example shown in Figure 7 because the notes that are common to both occurrences of the pattern do not all fall on strong beats. There is a multitude of string-processing algorithms available for discovering repeated factors in strings. However, there are far fewer algorithms available for finding repeated subsequences (i.e. patterns with gaps) and most of these seem to be NP-complete.

$$
\begin{array}{llll}
\{ & \langle 0,27,16,2,2\rangle, & \langle 1,46,27,1,1\rangle, & \langle 2,39,23,2,2\rangle, \\
 & \langle 2,44,26,1,1\rangle, & \langle 3,46,27,1,1\rangle, & \langle 4,32,19,2,2\rangle, \\
 & \langle 4,47,28,1,1\rangle, & \langle 5,44,26,1,1\rangle, & \langle 6,39,23,2,2\rangle, \\
 & \langle 6,42,25,1,1\rangle, & \langle 7,44,26,1,1\rangle, & \langle 8,30,18,2,2\rangle, \\
 & \langle 8,46,27,1,1\rangle, & \langle 9,42,25,1,1\rangle, & \langle 10,39,23,2,2\rangle, \\
 & \ldots & \ldots & \ldots \\
 & \langle 26,29,17,1,2\rangle, & \langle 26,51,30,2,1\rangle, & \langle 27,30,18,1,2\rangle, \\
 & \langle 28,32,19,1,2\rangle, & \langle 28,41,24,2,1\rangle, & \langle 29,29,17,1,2\rangle, \\
 & \langle 30,27,16,1,2\rangle, & \langle 30,50,29,2,1\rangle, & \langle 31,29,17,1,2\rangle \quad \}
\end{array}
$$

Figure 9: Five-dimensional dataset representing the music in Figure 8.

5 Representing music using multidimensional datasets

All the algorithms discussed in the previous section are based on the assumption that the music is represented either as a 1-dimensional string of symbols or, in the case of polyphonic music, as a set of such symbol strings and this assumption seems to be the cause of many of their short-comings. For example, the fact that they process symbol strings means that these algorithms cannot deal with unvoiced polyphonic music such as keyboard music. Their string-matching basis also causes problems when it comes to finding patterns that are distributed between several voices or finding transposed occurrences of polyphonic patterns with gaps.

We have avoided these problems by adopting a *geometric* approach in which the music is represented as a multidimensional dataset. A multidimensional dataset is just a finite set of position vectors or datapoints in a Euclidean space with a finite number of dimensions. Our algorithms work with datasets of any dimensionality and any size. Also the co-ordinates may take real values.

There are many possible appropriate ways of representing a piece of music as a multidimensional dataset. For example, Figure 9 shows a 5-dimensional dataset that represents the score in Figure 8. Each datapoint in Figure 9 represents a single note event in the score. The co-ordinate values in each datapoint represent onset time (in semiquavers), chromatic pitch (Meredith, 1999, 2001) (essentially MIDI number—a number indicating the key on a normal piano keyboard that would have to be pressed to produce a tone with that pitch), morphetic pitch (Meredith, 1999, 2001) (essentially the same as Brinkman's (1986) 'continuous name code'), duration and voice respectively.

We can then consider various orthogonal projections of such a dataset. For example, we could just consider the first two dimensions in the dataset in Figure 9 and get the projection in Figure 10 which tells us the chromatic pitch and onset time of each note.

If we consider just the first and third dimensions in Figure 9 we get the two-dimensional graph of morphetic pitch (diatonic pitch) against onset time shown in Figure 11. Note that some of the patterns that were only similar in Figure 10 are now identical because we are using a representation of diatonic pitch. It is often

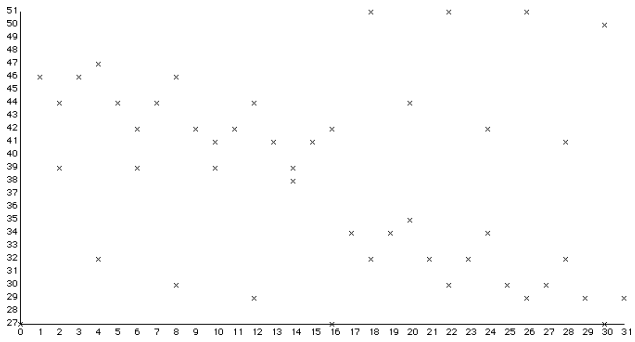

Figure 10: Two-dimensional projection of dataset in Figure 9 showing chromatic pitch against onset-time.

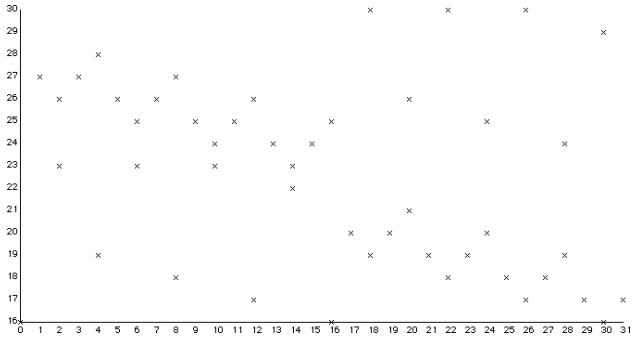

Figure 11: Two-dimensional projection of dataset in Figure 9 showing morphetic pitch against onset-time.

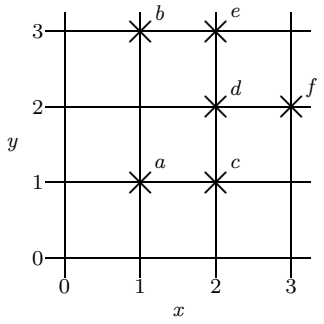

Figure 12: A simple dataset.

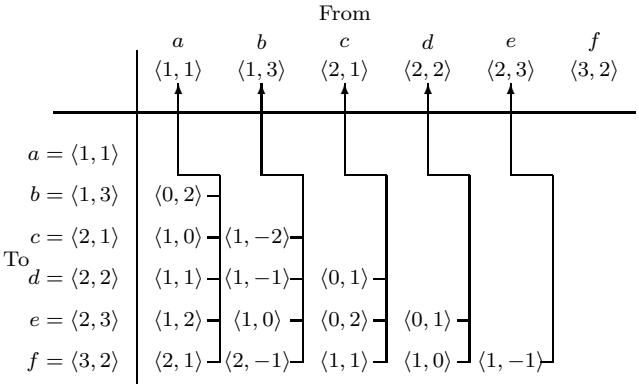

Figure 13: Vector table for dataset in Figure 12.

more profitable when analysing tonal music to look for exact repetitions in this type of projection than in the chromatic-pitch-against-onset-time representation.

Adopting this geometric approach allows us to find classes of perceptually significant musical repetition that are very difficult to compute using string-based approaches. It also allows us to process polyphonic music as simply and efficiently as monophonic music. It dispenses with the need for multiple representations because we can run the same repetition discovery algorithms on various orthogonal projections of a single, rich multidimensional dataset representation. It also allows us to discover repetitions in the dynamic, timbre and rhythmic structure of a piece as well as its pitch structure.

6 SIA: Discovering maximal repeated patterns in multidimensional datasets

SIA takes a multidimensional dataset as input and finds for every possible vector the largest pattern in the dataset that can be translated by that vector to give another pattern in the dataset. For example, if we consider the dataset in Figure 12, then the largest pattern that can be translated by the vector $\langle 1, 0\rangle$ is the pattern $\{a, b, d\}$ and the largest pattern that can be translated by the vector $\langle 1, 1\rangle$ is the pattern $\{a, c\}$. We say that a pattern is *translatable* by a vector if it can be translated by the vector to give another pattern in the dataset. And we say that the *maximal translatable pattern* or MTP for a vector is the largest pattern that can be translated by the vector to give another pattern in the dataset. SIA discovers all the non-empty MTPs in a dataset.

The first step in SIA is to sort the dataset to be analysed. Then the algorithm constructs a *vector table* for the dataset like the one in Figure 13 which shows the vector table constructed by SIA for the dataset in Figure 12. A cell in the vector table contains the vector *from* the datapoint at the head of the column of that cell *to* the datapoint at the head of the row for that cell.

SIA computes all the values in the vector table below the leading diagonal as shown in Figure 13. In other words, it computes for each datapoint all the vectors from that datapoint to every other datapoint in the dataset greater than it. Note that each of the vectors in the vector table (Figure 13) is stored with a pointer that points back to the "origin" datapoint for which it was computed (that is, the datapoint at the top of its column).

SIA then simply sorts the vectors in the vector table using a slightly modified version of merge sort that takes advantage of the fact that the columns in this table are already sorted. This results in a list like the one shown in Figure 14 which shows the result of sorting the vectors in the vector table in Figure 13. Note

Vector		Datapoint
$\langle 0,1\rangle$	→	$\langle 2,1\rangle$
$\langle 0,1\rangle$	→	$\langle 2,2\rangle$
$\langle 0,2\rangle$	→	$\langle 1,1\rangle$
$\langle 0,2\rangle$	→	$\langle 2,1\rangle$
$\langle 1,-2\rangle$	→	$\langle 1,3\rangle$
$\langle 1,-1\rangle$	→	$\langle 1,3\rangle$
$\langle 1,-1\rangle$	→	$\langle 2,3\rangle$
$\langle 1,0\rangle$	→	$\langle 1,1\rangle$
$\langle 1,0\rangle$	→	$\langle 1,3\rangle$
$\langle 1,0\rangle$	→	$\langle 2,2\rangle$
$\langle 1,1\rangle$	→	$\langle 1,1\rangle$
$\langle 1,1\rangle$	→	$\langle 2,1\rangle$
$\langle 1,2\rangle$	→	$\langle 1,1\rangle$
$\langle 2,-1\rangle$	→	$\langle 1,3\rangle$
$\langle 2,1\rangle$	→	$\langle 1,1\rangle$

Figure 14: The vectors in the vector table in Figure 13 sorted into increasing order.

that each vector in this list is still linked to the datapoint at the head of its column in the vector table. Simply reading off all the datapoints attached to the adjacent occurrences of a given vector in this list gives us the maximal translatable pattern for that vector. The complete set of non-empty maximal translatable patterns can be obtained simply by scanning the list once, reading off the attached datapoints and starting a new pattern each time the vector changes. Each box in the right-hand column of the list corresponds to a maximal translatable pattern. The most expensive step in this process is sorting the vectors which can be done in a worst-case running time of $O(kn^2 \log_2 n)$ for a k-dimensional dataset of size n. The space complexity of this version of the algorithm is $O(kn^2)$. However, Ukkonen *et al.* (2003) describe an implementation of the algorithm that uses only $O(n)$ working space.

7 SIATEC: Discovering all the occurrences for each maximal translatable pattern

SIATEC first generates all the maximal translatable patterns using a slightly modified version of SIA and then it finds all the occurrences of each MTP.

As explained in the previous section, SIA only computes the vectors below the leading diagonal in the vector table (see Figure 13). This is because the maximal translatable pattern for a vector $-v$ is the same as the pattern that you get by translating the maximal translatable pattern for v by the vector v itself. However, it turns out that by computing *all* the vectors in the vector table we can more efficiently discover all the occurrences of any given pattern within the dataset. So, for the dataset in Figure 12, SIATEC would actually compute the complete vector table as shown in Figure 15 and would then use the region below the leading diagonal in this table to compute the MTPs as in SIA. Before computing this vector table, the dataset is sorted so that the vectors increase as one descends a column and decrease as one moves from left to right along a row.

We know that a given column in the vector table (see Figure 15) contains all the vectors that the datapoint at the top of the column can be translated by to give another point in the dataset. Say we want to find all the occurrences of the pattern $\{a, c\}$ in Figure 12 which is the maximal translatable pattern in this dataset for the vector $\langle 1,1\rangle$. When we say that we want to "find all the occurrences" of a pattern, all we actually need to find is all the vectors that the pattern is translatable by. So, for example, the pattern $\{a, c\}$ is only translatable by the vectors $\langle 1,1\rangle$ and $\langle 0,2\rangle$. We know that the column of vectors under a contains all the vectors that the point a can be translated by; and we know that the column under c contains all the vectors that the point c can be translated by. So we know that the pattern $\{a, c\}$ can only be translated by the vectors that occur in both of these columns. In other words, to find the set of occurrences for a given pattern we simply have to find the intersection set of the columns headed by the datapoints in the pattern. By exploiting the orderedness of this table, we can find all the occurrences of a k-dimensional pattern of size m in a dataset of size n in a worst-case running time of $O(kmn)$. We know that the complete set of maximal translatable patterns is found by SIA simply by sorting the vectors below the leading diagonal in the vector table. If there are l such patterns and m_i is the size of the ith pattern then this implies

$$\sum_{i=1}^{l} m_i \leq \frac{n(n-1)}{2}.$$

So the overall worst-case running time of SIATEC is

$$O\left(\sum_{i=1}^{l} km_i n\right) \leq O\left(\frac{kn^2(n-1)}{2}\right).$$

The worst-case time complexity of the algorithm is therefore $O(kn^3)$ for a k-dimensional dataset of size n. The space complexity of the algorithm is $O(kn^2)$ if one stores the complete vector table. However, if one uses the techniqe described by Ukkonen *et al.* (2003), this worst-case space complexity can be reduced to $O(kn)$.

8 Running SIA and SIATEC on music data

SIA and SIATEC were implemented in C and run on a 500MHz Sparc machine on 52 datasets ranging in size from 6 to 3500 datapoints and in dimensionality from 2 to 5 dimensions. The graph in Figure 16 shows the

		From					
		$a = \langle 1,1\rangle$	$b = \langle 1,3\rangle$	$c = \langle 2,1\rangle$	$d = \langle 2,2\rangle$	$e = \langle 2,3\rangle$	$f = \langle 3,2\rangle$
	$a = \langle 1,1\rangle$	$\langle 0,0\rangle$	$\langle 0,-2\rangle$	$\langle -1,0\rangle$	$\langle -1,-1\rangle$	$\langle -1,-2\rangle$	$\langle -2,-1\rangle$
	$b = \langle 1,3\rangle$	$\langle 0,2\rangle$	$\langle 0,0\rangle$	$\langle -1,2\rangle$	$\langle -1,1\rangle$	$\langle -1,0\rangle$	$\langle -2,1\rangle$
	$c = \langle 2,1\rangle$	$\langle 1,0\rangle$	$\langle 1,-2\rangle$	$\langle 0,0\rangle$	$\langle 0,-1\rangle$	$\langle 0,-2\rangle$	$\langle -1,-1\rangle$
To	$d = \langle 2,2\rangle$	$\langle 1,1\rangle$	$\langle 1,-1\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle 0,-1\rangle$	$\langle -1,0\rangle$
	$e = \langle 2,3\rangle$	$\langle 1,2\rangle$	$\langle 1,0\rangle$	$\langle 0,2\rangle$	$\langle 0,1\rangle$	$\langle 0,0\rangle$	$\langle -1,1\rangle$
	$f = \langle 3,2\rangle$	$\langle 2,1\rangle$	$\langle 2,-1\rangle$	$\langle 1,1\rangle$	$\langle 1,0\rangle$	$\langle 1,-1\rangle$	$\langle 0,0\rangle$

Figure 15: Vector table computed by `SIATEC` for dataset in Figure 12.

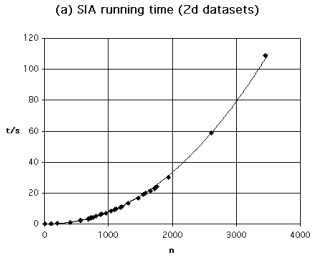

Figure 16: Running time of `SIA` on 2-dimensional datasets.

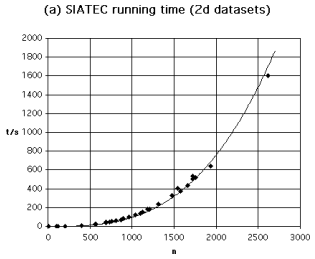

Figure 17: Running time of `SIATEC` on 2-dimensional datasets.

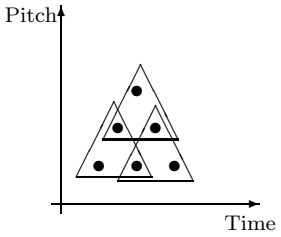

Figure 18: A dataset with three occurrences of a pattern indicated.

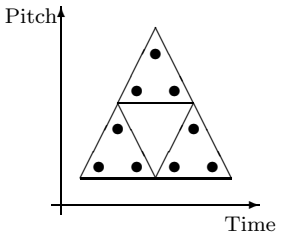

Figure 19: A dataset with three occurrences of a pattern indicated.

running time of `SIA` for the 2-dimensional datasets in the sample. The smooth curve in Figure 16 represents a running time of $kn^2 \log_2 n$. The graph in Figure 17 shows the running time of `SIATEC` on the same machine for the 2-dimensional datasets in the sample. In this graph, the smooth curve represents a running time of kn^3. As can be seen from these two graphs, it took less than 2 minutes for `SIA` to process a piece containing 3500 notes and about 13 minutes for `SIATEC` to process a piece containing 2000 notes.

9 Isolating significant repetitions

A dataset of size n contains 2^n distinct subsets and the number of patterns generated by `SIA` is less than $\frac{n^2}{2}$. Therefore, for all but the smallest datasets, `SIA` generates only a tiny fraction of all patterns in a dataset. Our experiments suggest that the repeated patterns that we're interested in (including many that are very hard to find using string-matching techniques) are often either equal to the maximal translatable patterns generated by `SIA` or straightforwardly derivable from them. Nevertheless, only a very small proportion of the patterns generated by `SIA` would be considered musically interesting by an analyst or expert listener. For example, `SIA` discovers over 70000 MTPs in Rachmaninoff's Prelude, Op.3 No.2 (see Figure 6) and probably fewer than 100 of these would be considered interesting by a music analyst. This means that we need to devise systems that evaluate the output of `SIA` and `SIATEC` and isolate various classes of musically interesting repetitions.

How can we isolate the theme-like and motive-like patterns in a passage? Some feasible heuristics for doing this will now be described.

First of all, it seems useful to define the *coverage* of a pattern to be the number of datapoints in the dataset that are members of occurrences of the pattern. For example, in Figure 18 the coverage of the triangular pattern is 6, but in Figure 19, the coverage of the triangular pattern is 9. Note that coverage is gen-

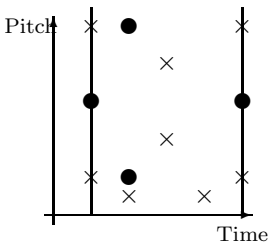

Figure 20: Defining 'the region spanned by a pattern' to be the time segment spanned by the pattern.

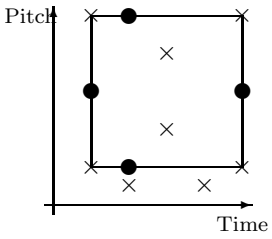

Figure 21: Defining 'the region spanned by a pattern' to be the bounding-box spanned by the pattern.

erally greater for patterns whose occurrences overlap less. It is also generally greater for larger patterns and for those that occur more often. In general, it seems that the most theme-like and motive-like patterns in music have relatively high coverage.

Next it seems useful to define the *compactness* of a pattern to be the ratio of the number of points in the pattern to the total number of points in the dataset that occur within the region spanned by the pattern within a particular representation. One can define 'the region spanned by a pattern' in a number of different ways. For example, we could define it to be the smallest time segment that contains the pattern, as illustrated in Figure 20. If we define it in this way, then the pattern consisting of the four round points in Figure 20 has a compactness value of $\frac{4}{12} = \frac{1}{3}$. Alternatively, one could define the region spanned by a pattern to be the bounding-box or the convex hull of the pattern in the pitch-against-onset-time graph of the piece. If we use the bounding box, as in Figure 21 then the compactness of the pattern consisting of the points marked by dots would be $\frac{4}{10} = \frac{2}{5}$. However, if we use the convex hull, as shown in Figure 22 then the compactness value of the same pattern in the same dataset would be $\frac{4}{6} = \frac{2}{3}$. Typically, at least one occurrence of a theme-like pattern will have a high compactness value, even if the other occurrences are highly embellished. Another interesting heuristic that seems to be useful for isolating theme-like patterns is the compression ratio that can be achieved by representing the set of points covered by all occurrences of the pattern by specifying simply one occurrence of the pattern and all the vectors by which the pattern can be translated. For example, in Figure 18 the set of points covered by the occurrences of the triangular pattern can be represented by specifying the points in one occurrence of the pattern and the two translation vectors that map that occurrence onto the other two occurrences of the pattern in this dataset. That is, we can represent the 6 datapoints in Figure 18 using 3 position vectors and 2 translation vectors, thus achieving a compression ratio of $\frac{6}{5}$. If we represent the dataset in Figure 19 in the same way we can achieve a compression ratio of $\frac{9}{5}$. In general, it seems that a high compression ratio can be achieved if we represent the set of points covered by all the occurrences of an important thematic pattern by specifying just one occurrence of the pattern and all the vectors by which the pattern can be translated. In the next section we use this idea of compression ratio in a compression algorithm based on `SIATEC` that can be used for generating analyses of pieces that in some cases resemble the thematic/motivic analyses carried out by human music analysts.

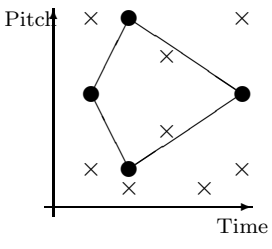

Figure 22: Defining 'the region spanned by a pattern' to be the convex hull spanned by the pattern.

10 `COSIATEC`: a compression algorithm based on `SIATEC`

`COSIATEC` is a compression algorithm based on `SIATEC`. The flow-chart in Figure 23 describes how it works. First we run `SIATEC` on the dataset to be compressed. This generates a list of $\langle$PATTERN, TRANSLATOR SET$\rangle$ pairs. The translator set for each pattern contains all the vectors by which the pattern is translatable within the dataset apart from the zero vector. In general, this gives a more efficient representation of the set of points covered by the occurrences of the pattern. Then the heuristics described in the previous section—compression ratio, coverage and compactness—are used to choose the 'best' pattern P and this pattern is printed out together with its translator set. Then all the points covered by P—that is all the points that are members of occurrences of P—are removed from the dataset. Then if the dataset is empty the algorithm terminates. However, if there are still datapoints in the dataset, `SIATEC` is again run on the remaining points and the cycle is repeated. The result is a print out of the 'best' pat-

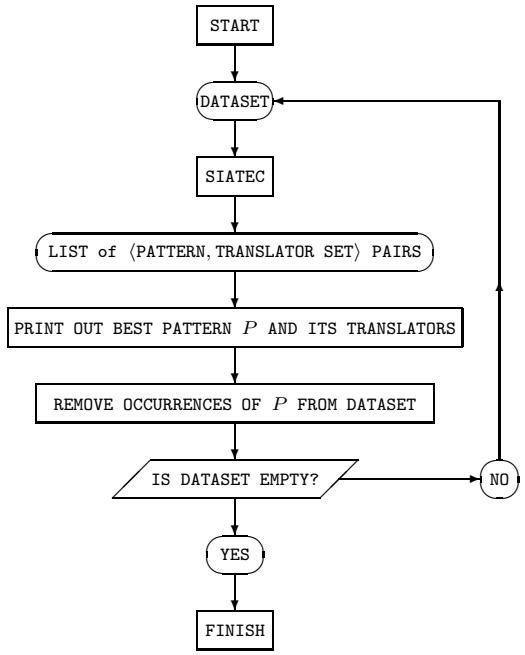

Figure 23: COSIATEC

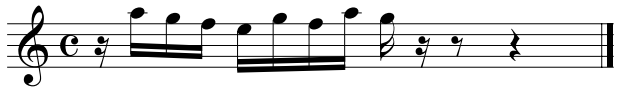

Figure 24: Pattern discovered on first iteration of COSIATEC when run on Bach's Two-Part Invention in C major, BWV 772.

tern and its translators for each iteration of the cycle, and this printout is, in general, a compressed representation of the input dataset. Obviously, the degree of compression achieved depends directly on the amount of repetition in the dataset.

COSIATEC has been run on morphetic-pitch-against-onset-time representations of all 15 of Bach's Two-part Inventions and the results are quite encouraging. In particular, it seems that the patterns that achieve the highest compression ratios on the early iterations of COSIATEC quite often correspond to the most important themes and motives in the music.

For example, Figures 24 and 25 show the patterns discovered by COSIATEC on the first and second iterations, respectively, when the algorithm is run on the Two-Part Invention in C major, BWV 772. The pattern in Figure 24 is the inversion of the subject and the pattern in Figure 25 is the subject itself. These

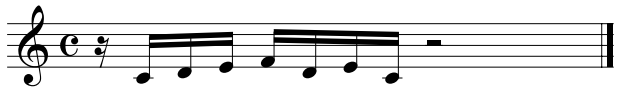

Figure 25: Pattern discovered on second iteration of COSIATEC when run on Bach's Two-Part Invention in C major, BWV 772.

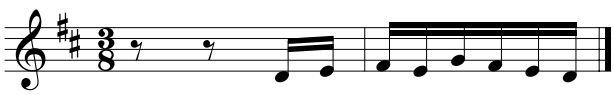

Figure 26: Pattern discovered on first iteration of COSIATEC when run on Bach's Two-Part Invention in D major, BWV 774.

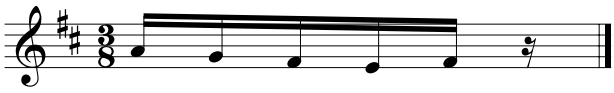

Figure 27: Pattern discovered on second iteration of COSIATEC when run on Bach's Two-Part Invention in D major, BWV 774.

two patterns are precisely the ones identified by Boyd (1983, p. 96) to be the two most important motivic patterns in this piece.

When COSIATEC is run on the Two-Part Invention in D major, BWV 774, the pattern that emerges on the first iteration consists of the anacrusis and first bar of the subject (Figure 26) and the pattern that is found on the second iteration consists of the second part of the subject (Figure 27). Similarly, when COSIATEC is run on the Invention in D minor, BWV 775, the subject of the invention is generated on the first iteration (Figure 28).

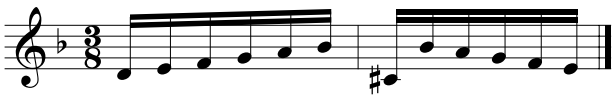

Figure 28: Pattern discovered on first iteration of COSIATEC when run on Bach's Two-Part Invention in D minor, BWV 775.

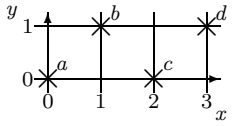

Figure 29: Example input query pattern for SIAMESE.

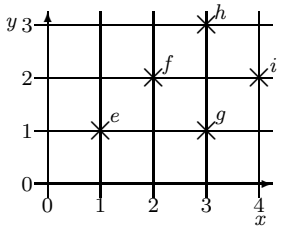

Figure 30: Example input dataset for SIAMESE.

11 SIAMESE: Pattern matching in multidimensional datasets

We have also developed a pattern-matching algorithm based on SIA which we call SIAMESE. This algorithm takes a multidimensional query pattern and a multidimensional dataset as input and finds all exact complete and partial matches of the query pattern in the dataset. For example, if we run SIAMESE on the query pattern in Figure 29 and the dataset in Figure 30 then the output will tell us (amongst other things) that the complete query pattern $\{a, b, c, d\}$ can be matched to the pattern $\{e, f, g, i\}$ in the dataset. It will also tell us that the three point pattern $\{a, b, c\}$ in the query can be matched to the pattern $\{f, h, i\}$ in the dataset, that the points $\{c, d\}$ can be matched to $\{e, f\}$ and $\{f, h\}$ and so on.

SIAMESE works in essentially the same way as SIA. We begin by sorting the points in the query and the points in the dataset and then we construct a vector table like the one in Figure 31 which shows the vector table constructed for the query pattern in Figure 29 and the dataset in Figure 30. Each entry in this table gives the vector *from* the query datapoint at the head of the colum in which the entry occurs *to* the dataset point at the head of the row in which the entry occurs. Note that as in SIA, each vector in the vector table has a pointer that points back to the query pattern datapoint at the head of the column in which it occurs.

	From a $\langle 0,0\rangle$	b $\langle 1,1\rangle$	c $\langle 2,0\rangle$	d $\langle 3,1\rangle$
$e = \langle 1,1\rangle$	$\langle 1,1\rangle$	$\langle 0,0\rangle$	$\langle -1,1\rangle$	$\langle -2,0\rangle$
$f = \langle 2,2\rangle$	$\langle 2,2\rangle$	$\langle 1,1\rangle$	$\langle 0,2\rangle$	$\langle -1,1\rangle$
To $g = \langle 3,1\rangle$	$\langle 3,1\rangle$	$\langle 2,0\rangle$	$\langle 1,1\rangle$	$\langle 0,0\rangle$
$h = \langle 3,3\rangle$	$\langle 3,3\rangle$	$\langle 2,2\rangle$	$\langle 1,3\rangle$	$\langle 0,2\rangle$
$i = \langle 4,2\rangle$	$\langle 4,2\rangle$	$\langle 3,1\rangle$	$\langle 2,2\rangle$	$\langle 1,1\rangle$

Figure 31: The vector table computer by SIAMESE for the query patter in Figure 29 and the dataset in Figure 30.

Having constructed this table, we then simply sort all the vectors in it to give a list like the one in Figure 32 which shows the list that results when the vectors in the table in Figure 31 are sorted. This list gives us all the vectors that we can translate the query pattern by to give a non-empty match in the dataset. The fact that each of these vectors still has a pointer to the query pattern datapoint at the head of its column in the vector table means that, for each vector, we can simply read off the points in the query pattern that have matches in the dataset when the query pattern is translated by that vector. For example, if we look at the query pattern datapoints that are pointed to by vectors with the value $\langle 1,1\rangle$ in Figure 32, we find that all four of the points in the query pattern are matched which tells us that a complete occurrence of the query pattern occurs at a displacement of $\langle 1,1\rangle$. Similarly, if we look at the datapoints attached to the consecutive occurrences of the vector $\langle 2,2\rangle$ in this list, we find that the points $\{a, b, c\}$ are matched when the query is translated by this vector.

VECTOR	DATAPOINT
$\langle -2,0\rangle$	$\langle 3,1\rangle$
$\langle -1,1\rangle$	$\langle 2,0\rangle$
$\langle -1,1\rangle$	$\langle 3,1\rangle$
$\langle 0,0\rangle$	$\langle 1,1\rangle$
$\langle 0,0\rangle$	$\langle 3,1\rangle$
$\langle 0,2\rangle$	$\langle 2,0\rangle$
$\langle 0,2\rangle$	$\langle 3,1\rangle$
$\langle 1,1\rangle$	$\langle 0,0\rangle$
$\langle 1,1\rangle$	$\langle 1,1\rangle$
$\langle 1,1\rangle$	$\langle 2,0\rangle$
$\langle 1,1\rangle$	$\langle 3,1\rangle$
$\langle 1,3\rangle$	$\langle 2,0\rangle$
$\langle 2,0\rangle$	$\langle 1,1\rangle$
$\langle 2,2\rangle$	$\langle 0,0\rangle$
$\langle 2,2\rangle$	$\langle 1,1\rangle$
$\langle 2,2\rangle$	$\langle 2,0\rangle$
$\langle 3,1\rangle$	$\langle 0,0\rangle$
$\langle 3,1\rangle$	$\langle 1,1\rangle$
$\langle 3,3\rangle$	$\langle 0,0\rangle$
$\langle 4,2\rangle$	$\langle 0,0\rangle$

Figure 32: List of sorted vectors generated by SIAMESE for the query pattern in Figure 29 and the dataset in Figure 30.

The most expensive step in this process is sorting the vectors in the vector table. Using a comparison sort such as merge sort, this step can be achieved in a worst-case running time of $O(knm \log_2(mn))$ for a k-dimensional query pattern of size m and a k-dimensional dataset of size n.

12 Possible directions for further work

We finish by suggesting some directions in which further work could be carried out.

- Versions of SIA and SIATEC, COSIATEC and SIAMESE need to be developed that discover *approximate* repetitions (see Ukkonen *et al.*, 2003).
- Algorithms (possibly based on SIA) could be developed for discovering repetitions where the patterns are related by rotation, reflection or dilatation (enlargement).

- The running times of `SIA` and `SIATEC` could be improved by using word parallelism or by designing PRAM versions of the algorithms.
- Further heuristics and algorithms could be developed for isolating various classes of perceptually significant repetition.
- Applications of the algorithms could be developed for use in specific domains (e.g., music, images, video, bioinformatics):
  - `SIA`, `SIATEC` and `COSIATEC` can be used for
    - data compression,
    - database indexing and
    - data mining.
  - `SIAMESE` can be used for
    - information retrieval and
    - computer-based learning systems.

References

Bent, I. and Drabkin, W. (1987). *Analysis*. New Grove Handbooks in Music. Macmillan.

Boyd, M. (1983). *Bach*. The Master Musicians. J. M. Dent, London.

Brinkman, A. R. (1986). A binomial representation of pitch for computer processing of musical data. *Music Theory Spectrum*, **8**, 44–57.

Cambouropoulos, E. (1998). *Towards a General Computational Theory of Musical Structure*. Ph.D. thesis, University of Edinburgh.

Conklin, D. and Anagnostopoulou, C. (2001). Representation and discovery of multiple viewpoint patterns. In *Proceedings of the International Computer Music Conference, 2001, Havana Cuba*.

Crochemore, M. (1981). An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, **12**(5), 244–250.

Forte, A. and Gilbert, S. E. (1982). *Introduction to Schenkerian Analysis*. Norton, New York.

Hsu, J.-L., Liu, C.-C., and Chen, A. L. (1998). Efficient repeating pattern finding in music databases. In *Proceedings of the 1998 ACM 7th International Conference on Information and Knowledge Management*, pages 281–288. Association of Computing Machinery.

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

Meredith, D. (1999). The computational representation of octave equivalence in the Western staff notation system. In *Cambridge Music Processing Colloquium, September 1999.* Available online at `http://www-sigproc.eng.cam.ac.uk/music_proc/submissions/`.

Meredith, D. (2001). MIPS: A formal language for the mathematical investigation of pitch systems (version 2001-09-10). Available online at `http://www.titanmusic.com/papers.html`.

Meredith, D., Lemström, K., and Wiggins, G. A. (2002a). Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*. To appear. (Draft available online at `http://www.titanmusic.com/papers.html`).

Meredith, D., Wiggins, G. A., and Lemström, K. (2002b). Method of pattern discovery. PCT patent application number PCT/GB02/02430, UK patent application number 0211914.7. Applied for by City University, London and filed on 23 May 2002. (Priority date: 23 May 2001, draft available online at `http://www.titanmusic.com/papers.html`).

Nattiez, J.-J. (1975). *Fondements d'une sémiologie de la musique*. Union Générale d'Éditions, Paris.

Rolland, P.-Y. (1999). Discovering patterns in musical sequences. *Journal of New Music Research*, **28**(4), 334–350.

Ruwet, N. (1972). *Langage, Musique, Poésie*. Éditions du seuil, 27, rue Jacob, Paris VI.

Schenker, H. (1954). *Harmony*. University of Chicago Press, London. Edited by Oswald Jonas and translated by Elisabeth Mann Borgese from the 1906 German edition.

Temperley, D. (2001). *The Cognition of Basic Musical Structures*. MIT Press, Cambridge, MA.

Ukkonen, E., Lemström, K., and Mäkinen, V. (2003). Sweepline the music! In R. Klein, H.-W. Six, and L. Wegner, editors, *Computer Science in Perspective (LNCS 2598)*, pages pp. 330–342.