Point-set algorithms for pattern discovery and pattern matching in music

David Meredith

Goldsmiths College, University of London, Department of Computing
New Cross, London, SE14 6NW.
dave@titanmusic.com

Abstract. An algorithm that discovers the themes, motives and other perceptually significant repeated patterns in a musical work can be used, for example, in a music information retrieval system for indexing a collection of music documents so that it can be searched more rapidly. It can also be used in software tools for music analysis and composition and in a music transcription system or model of music cognition for discovering grouping structure, metrical structure and voice-leading structure. In most approaches to pattern discovery in music, the data is assumed to be in the form of strings. However, string-based methods become inefficient when one is interested in finding highly embellished occurrences of a query pattern or searching for polyphonic patterns in polyphonic music. These limitations can be avoided by representing the music as a set of points in a multidimensional Euclidean space. This point-set pattern matching approach allows the maximal repeated patterns in a passage of polyphonic music to be discovered in quadratic time and all occurrences of these patterns to be found in cubic time. More recently, Clifford *et al.* [1] have shown that the best match for a query point set within a text point set of size n can be found in $O(n \log n)$ time by incorporating randomised projection, uniform hashing and FFT into the point-set pattern matching approach. Also, by using appropriate heuristics for selecting compact maximal repeated patterns with many non-overlapping occurrences, the point-set pattern discovery algorithms described here can be adapted for data compression. Moreover, the efficient encodings generated when this compression algorithm is run on music data seem to resemble the motivic-thematic analyses produced by human experts.

Keywords. Content-based music information retrieval, point-set pattern matching

1 Introduction

An algorithm that discovers the themes, motives and other perceptually significant and memorable repeated patterns in a musical work can be used, for example, in a music information retrieval system for indexing a collection of music documents so that it can be searched more rapidly. Perhaps less obviously, such an algorithm can also be used to aid with metrical structure and

Dagstuhl Seminar Proceedings 06171
Content-Based Retrieval
http://drops.dagstuhl.de/opus/volltexte/2006/652

grouping structure analysis, since the different occurrences of a significant repeated pattern usually occur at corresponding positions within a bar or group [2, pp. 51, 75]. A pattern discovery algorithm could also be used in a tool for assisting composition to suggest ways in which the patterns in the music already composed may be developed further.

A number of influential music analysts and music psychologists have stressed that discovering the important repetitions in a passage of music is an essential step towards achieving a rich understanding of it. For example, Heinrich Schenker claimed that repetition "is the basis of music as an art" [3, p. 5], Ian Bent proposed that "the central act" in all forms of music analysis is "the test for identity" [4, p. 5] and Lerdahl and Jackendoff [2, p. 52] state that

> the importance of parallelism [i.e., repetition] in musical structure cannot be overestimated. The more parallelism one can detect, the more internally coherent an analysis becomes, and the less independent information must be processed and retained in hearing or remembering a piece.

However, although detecting the significant repetitions in a passage of music is essential if one wishes to understand it, the vast majority of the repeated patterns that occur in a typical musical passage are neither intended by the composer nor perceived by the listener. Consider, for example, Figure 1 which shows the first six bars of Rachmaninoff's *Prelude* in C♯ minor, Op. 3, No. 2. In this passage, the listener is obviously intended to hear that bar 4 is a repetition of bar 3, that bars 5 and 6 are modified repetitions of bar 3 and that the descending bass figure in bars 1-2 is repeated in bars 3-4, 4-5 and 5-6. However, it seems very unlikely that Rachmaninoff intended us to hear that the pattern consisting of the notes in circles is repeated 7 crotchets later and transposed up a minor 9th to give the pattern consisting of the notes in squares.

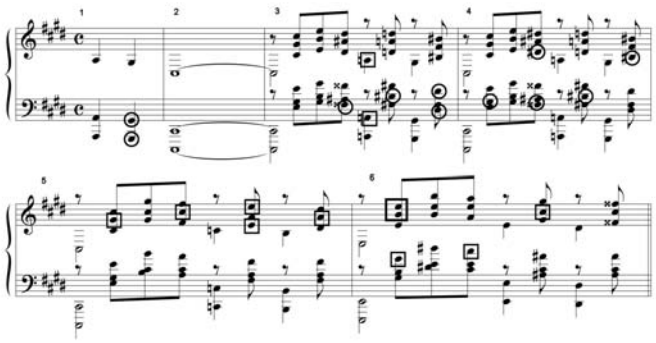

Fig. 1. Example of a repeated pattern that is neither intended by the composer nor heard by the listener. (From the beginning of Rachmaninoff's *Prelude* in C♯ minor, Op. 3, No. 2.)

To be a useful tool and a plausible model of music cognition, a computer program for discovering the significant repetitions in a passage of music should be able to find all and only those repetitions that an expert listener or music analyst would consider to be important and interesting. Unfortunately, there are at least two reasons why this class of "interesting" repetitions is very diverse. First, the patterns that are involved in such "interesting" repetitions vary widely in their structural characteristics; and, second, there are many ways of transforming a musical pattern to give another pattern that is perceived to be a version of it (for example, it can be truncated, augmented, diminished, inverted, embellished or even reversed).

I shall now present a couple of examples that illustrate the diversity of the class of interesting musical repetitions. An interesting repeated pattern may be a small motive, consisting of just a few notes, or a whole section of a symphonic movement containing hundreds or even thousands of notes. Figure 2 shows an example of a small (but important) motive occurring five times (A1–5) in the first few bars of Barber's *Sonata for Piano*, Op. 26. There are a number of points to note about this motive. First, it only contains notes in the left hand part and, as it is being played, the right hand part plays a different repeated pattern whose period is different from that of the bass motive. Second, the different occurrences of the bass motive are not all at the same transposition (patterns A4–5 are a perfect fourth higher than A1–3). Third, the duration of the final octave dyad in each occurrence of the pattern is not always the same. Finally, the five occurrences are consecutive—that is, nothing happens in the bass part in between the occurrences of this motive.

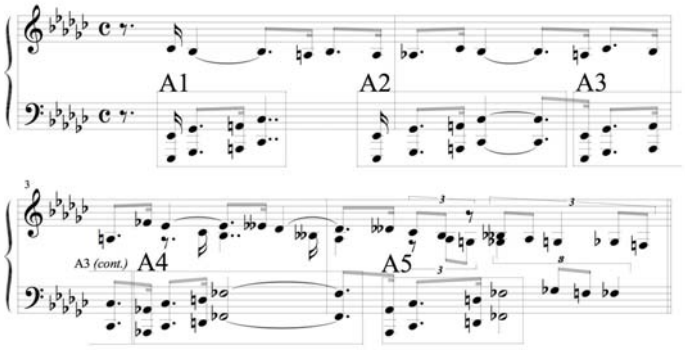

Fig. 2. Five consecutive occurrences of a small bass motive in the first few bars of Barber's *Sonata for Piano*, Op. 26.

Now consider Figure 3, which shows the first few bars of *Contrapunctus VI* from J. S. Bach's *Die Kunst der Fuge* (BWV 1080). This example shows a theme being transformed by the musical transformations of diminution, transposition

and inversion. The alto part is derived from the bass part by transposing it up an octave, delaying its start for two and a half bars and halving all the note durations. If we represented the notes in the music as points in a graph of pitch against time, then the bass part would be mapped onto the alto part by translation and then scaling. The soprano part is an inversion of the alto part. So, in geometric terms, to obtain the soprano part from the bass part, one would have to reflect the bass part in a horizontal axis and then scale by a factor of 1/2 parallel to this axis. Note that the three occurrences of the subject overlap in both time and pitch. Therefore, whereas the patterns in Figure 2 were "bounding-box" compact, the patterns in Figure 3 are compact only in the sense that each occurrence contains all the notes that occur between two time points within a particular voice.

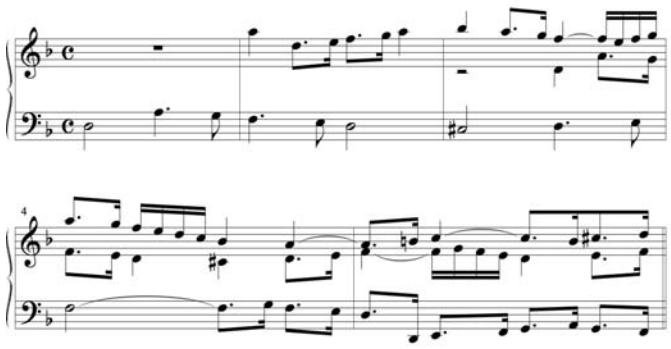

Fig. 3. The opening bars of J. S. Bach's *Contrapunctus VI* from *Die Kunst der Fuge* (BWV 1080).

2 Problems with the string-based approach to discovering and matching patterns in music

In many previous approaches to discovering and matching patterns in musical data, it has been assumed that the music is represented as a set of strings, with each string representing a voice or part and each symbol within each string representing a note or an interval between two consecutive notes [5,6,7,8,9]. Also, the similarity between two musical patterns, represented as strings, has usually been measured in terms of the edit distance between them, calculated using dynamic programming [5,8].

A problem arises, however, when one attempts to use this string-based approach for finding highly embellished variations on a query pattern. Consider, for example, the two patterns, A and B, shown in Figure 4. B is clearly an embellished variation on the rising arpeggio, A. One might therefore want a pattern

discovery or pattern matching algorithm to identify A and B as being occurrences of the same pattern. However, if A and B are represented as strings in which each symbol represents the pitch of a note, then the edit distance between A and B is 9 which is rather large, relative to the sizes of the patterns involved. In order to recognize B as a match for A, an edit-distance-based algorithm would therefore have to allow patterns differing by up to 9 edit operations to count as matches. However, this would probably result in many spurious matches since there could be many patterns differing by 9 edit operations from A that are not perceived to be similar to A.

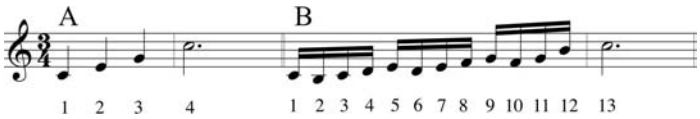

Fig. 4. Pattern B is an embellished variation on pattern A but the edit distance between A and B is 9 which is large relative to the sizes of the patterns involved.

Another problem with the string-based approach to musical pattern matching arises when we want to search for patterns in polyphonic music. Specifically, if we do not know the voice to which each note belongs (e.g., in much piano music) or if we are interested in patterns containing notes from two or more voices, then the number of passage-length strings required to represent the passage fully is exponential in the number of distinct onsets. Moreover, if we don't search through *all* of these strings, then we run the risk of missing certain occurrences of a query. This is illustrated in Figure 5 which shows a graph of pitch against time representing the beginning of the folk-song *Frère Jacques*. Each point in this figure represents a note and each pair of notes that could be consecutive within a single string are joined by an arrow. Note that the number of strings required is multiplied by k each time a note is encountered which is followed by k notes that start simultaneously.

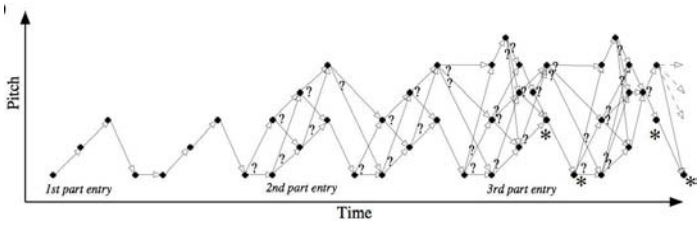

Fig. 5. Representing fully a passage of unvoiced polyphonic music produces a combinatorial explosion in the number of strings required. (Figure provided by Geraint Wiggins.)

3 Using multidimensional point sets to represent music

The problems described in the previous section can be avoided by using multidimensional point sets instead of strings to represent music and then developing algorithms to work on these point sets. Figure 7 shows one useful way in which the two-bar passage in Figure 6 could be represented as a multidimensional point set. Each 5-tuple in Figure 7 represents a single note. The first element in each 5-tuple gives the onset time of the note. The second element gives the chromatic pitch which is just 21 less than the MIDI note number of the note. The third element in each 5-tuple gives the diatonic pitch which is an integer that indicates the position of the note-head of the note on the staff. The fourth element indicates the duration of the note and the fifth indicates the voice to which the note belongs.

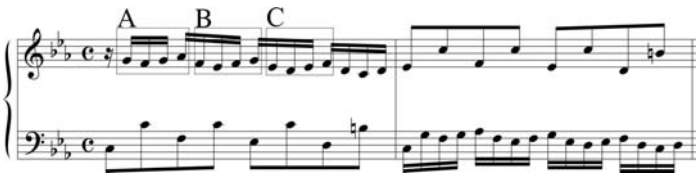

Fig. 6. The first two bars of J. S. Bach's *Prelude* in C minor (BWV 871).

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

Fig. 7. A five-dimensional point set representation of the music in Figure 6. See text for explanation.

When searching for patterns or attempting to discover repetitions, however, it is usually more useful to use a 2- or 3-dimensional projection of such a multidimensional point set, such as the ones shown in Figures 8 and 9. In Figure 8, each point gives the onset time and chromatic pitch of a note and the patterns A, B and C in Figure 6 are indicated by rectangles. Note that the three patterns, A, B and C, clearly sound like versions of the same motive, even though, as shown

in Figure 8, their chromatic interval structures are different from each other. One would therefore have to use an approximate matching algorithm to match these three patterns in the projection shown in Figure 8.

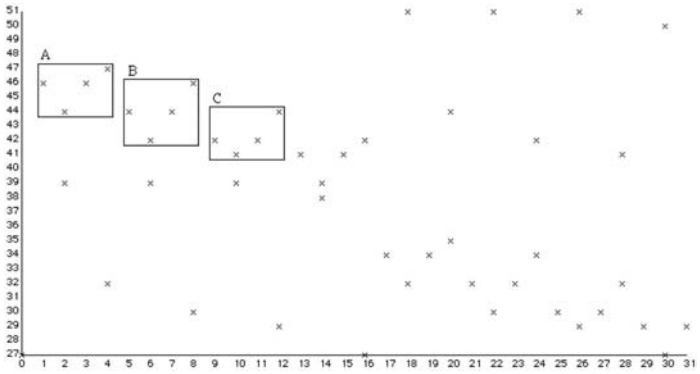

Fig. 8. A 2-dimensional projection of the point set in Figure 7 showing the onset time and chromatic pitch of each note.

However, in the musical notation in Figure 6, the pitch names of the notes are carefully chosen so that the similarity between patterns A, B and C is nicely represented by the fact that they all have the same scale-step structure (i.e., a descending step followed by two ascending steps). This can be exploited by using a 2-dimensional projection of the point set in Figure 7 in which each point gives the onset time and *diatonic* pitch of the note (see Figure 9). When this is done, the three patterns have translation-invariant representations and can thus be found using a fast, exact-matching algorithm. When analysing tonal music, a projection like the one in Figure 9 is usually more useful than one like Figure 8. The process of computing a diatonic pitch representation like the one in Figure 9 from a chromatic pitch representation like the one in Figure 8 is called *pitch spelling*. A number of highly effective algorithms have been developed for carrying out this process (see [10,11]).

I shall now present a number of algorithms for pattern discovery and pattern matching in multidimensional point sets that can fruitfully be applied to music data.

4 SIA: Discovering all the maximal translatable patterns (MTPs) in a multidimensional point set

The first point-set algorithm to be discussed here is SIA (which stands for Structure Induction Algorithm) [12]. This algorithm takes as input a multidimensional point set and discovers, for every vector, the points in this point set that are mapped onto other points in the point set by that vector. For convenience, let

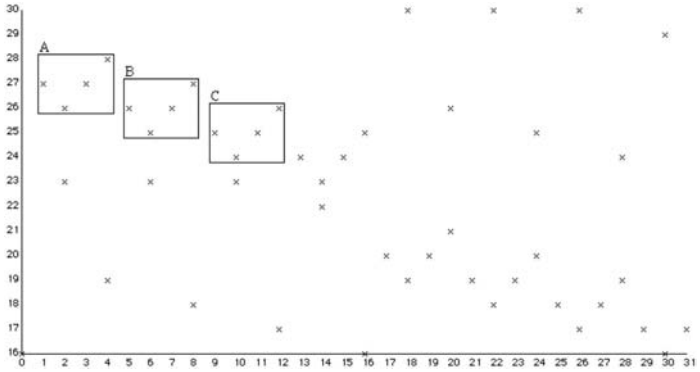

Fig. 9. A 2-dimensional projection of the point set in Figure 7 showing the onset time and diatonic pitch of each note.

us call the complete set of points being analysed the *dataset* and let a *pattern* be any subset of this dataset. We say that a pattern is *translatable* by a particular vector within a dataset if it can be translated by that vector to give another pattern in the dataset. For example, the pattern $\{a, d\}$ in the dataset in Figure 10 is translatable within this dataset by the vector $\langle 1, 0\rangle$. The pattern that contains all the points in a dataset that are mapped by a particular vector onto other points in the dataset is called the *maximal translatable pattern* (MTP) for that vector in that dataset. For example, the MTP for the vector $\langle 1, 0\rangle$ in the dataset in Figure 10 is $\{a, b, d\}$ and the MTP for the vector $\langle 1, 1\rangle$ in this dataset is $\{a, c\}$.

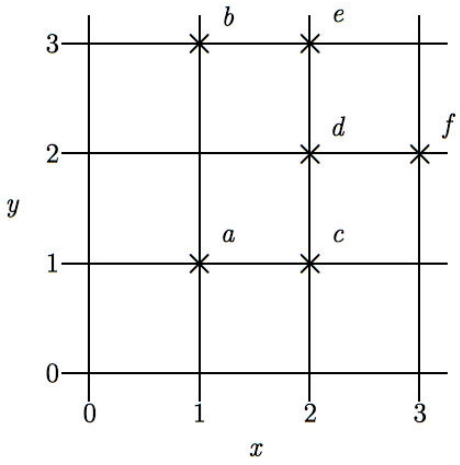

Fig. 10. A 2-dimensional dataset containing 6 points.

SIA discovers all the non-empty MTPs in a dataset. It does this by first sorting the points in the dataset into lexicographical order and then computing the vector from each point to each lexicographically greater point. These vectors are then stored in a table like the one shown in Figure 11. Each vector is stored with a pointer to the origin point for which it was computed, as indicated by the arrows in Figure 11.

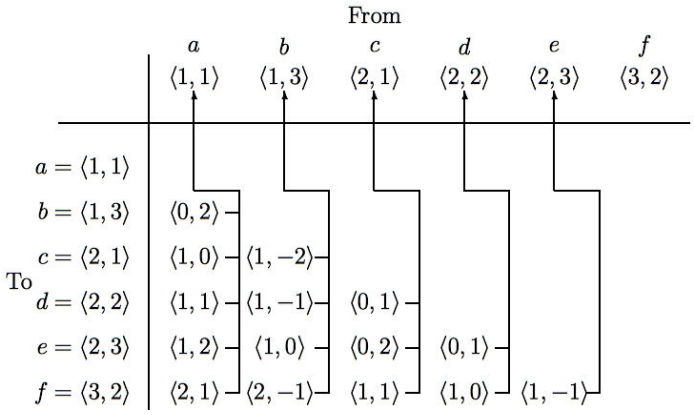

Fig. 11. A vector table storing the vector from each point to each lexicographically greater point in the dataset in Figure 10. Note that each vector is stored with a pointer to the origin point for which it was computed.

The vectors in this table are then sorted into lexicographical order to give a list like the one shown in Figure 12. The MTP for any vector can be found simply by reading off the origin points attached to the adjacent occurrences of that vector in this list. All the non-empty MTPs can therefore be generated by scanning this list once, printing out the origin datapoint attached to each vector and starting a new MTP each time the vector changes. The most expensive step in this algorithm is sorting the vectors which can be done in $O(kn^2 \log n)$ time in the worst case for a k-dimensional dataset containing n points. However, by storing the origin points in a hash table and hashing the vectors to get the slot indices, this time complexity can be improved to $O(kn^2)$ on average. The algorithm uses $O(kn^2)$ space.

5 SIATEC: Discovering all the translationally invariant occurrences of all the MTPs in a point set

Let's define the *translational equivalence class* (TEC) of a pattern in a dataset to be the set of all the translationally invariant occurrences of that pattern in the dataset. For example, the TEC that contains the pattern $\{a, b, d\}$ in Figure 10

Vector		Datapoint
⟨0, 1⟩	→	⟨2, 1⟩
⟨0, 1⟩	→	⟨2, 2⟩
⟨0, 2⟩	→	⟨1, 1⟩
⟨0, 2⟩	→	⟨2, 1⟩
⟨1, −2⟩	→	⟨1, 3⟩
⟨1, −1⟩	→	⟨1, 3⟩
⟨1, −1⟩	→	⟨2, 3⟩
⟨1, 0⟩	→	⟨1, 1⟩
⟨1, 0⟩	→	⟨1, 3⟩
⟨1, 0⟩	→	⟨2, 2⟩
⟨1, 1⟩	→	⟨1, 1⟩
⟨1, 1⟩	→	⟨2, 1⟩
⟨1, 2⟩	→	⟨1, 1⟩
⟨2, −1⟩	→	⟨1, 3⟩
⟨2, 1⟩	→	⟨1, 1⟩

Fig. 12. The list that results when the vectors in the table in Figure 11 are sorted into lexicographical order. Note that the MTP for any given vector can be found by reading off the origin points attached to the adjacent occurrences of that vector in this list.

is $\{\{a, b, d\}, \{c, e, f\}\}$. A TEC can be specified in a compact form as an ordered pair, $\langle P, V \rangle$, in which P is one pattern occurrence in the TEC and V is the set of non-zero vectors that map that occurrence onto other occurrences in the TEC. For example, the TEC $\{\{a, b, d\}, \{c, e, f\}\}$ in the dataset in Figure 10 could be represented as the ordered pair $\langle \{a, b, d\}, \{\langle 1, 0 \rangle\} \rangle$.

The algorithm SIATEC (which stands for "Structure Induction Algorithm that finds TECs") finds all the non-empty TECs in a multidimensional point set [12]. It does this by first computing a vector table like the one shown in Figure 13. It then uses the elements below the leading diagonal in this table to compute all the MTPs in the same way as SIA. The TEC for each MTP can then be computed by finding the intersection of the columns in this table that are headed by points in the MTP. For example, the MTP in the dataset in Figure 10 for the vector $\langle 1, 1 \rangle$ is $\{a, c\}$, and this point set is translatable by the vectors $\{\langle 0, 0 \rangle, \langle 0, 2 \rangle, \langle 1, 1 \rangle\}$, which is the set intersection of the columns headed by points a and c in Figure 13. The TEC for this MTP can therefore be represented by the ordered pair $\langle \{a, c\}, \{\langle 0, 2 \rangle, \langle 1, 1 \rangle\} \rangle$.

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

Fig. 13. The table of vectors computed by SIATEC for the dataset in Figure 10.

Finding all the vectors by which a pattern of size m in a k-dimensional dataset of size n is translatable takes $O(kmn)$ time if we do this by computing the intersection of the columns in the vector table headed by the points in the pattern. Let's suppose that the number of MTPs computed using SIA for a dataset of size n is ℓ and that the size of each of these MTPs is m_i, $1 \leq i \leq \ell$. From Figures 11 and 12, it is obvious that

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

Therefore, the time taken by SIATEC to compute all the occurrences of all the MTPs (after the MTPs have been found using SIA) is

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

Therefore, the worst-case running time of SIATEC is $O(kn^3)$ and its worst-case space complexity is $O(kn^2)$.

6 Experimental running times of SIA and SIATEC

SIA and SIATEC were implemented in C and run on a 500MHz Sparc on 52 datasets ranging in size from 6 to 3456 points. The dimensionality of the datasets ranged from 2 to 5. Figures 14 and 15 show the results of this experiment for SIA and SIATEC, respectively. As can be seen in these graphs, the observed running times were in close agreement with the calculated time complexities. It took less than 2 minutes for SIA to process a piece containing 3500 notes and around 13 minutes for SIATEC to process a piece with 2000 notes.

7 Using heuristics to isolate the interesting MTPs

A dataset of size n contains 2^n distinct subsets whereas SIA generates fewer than $n^2/2$ patterns. This implies that, for a passage of music containing at least a few hundred notes, SIA selects and generates only a very small fraction of all the patterns in the piece. It has also been found that many interesting patterns are either found by SIA or straightforwardly derivable from patterns found by SIA. However, many of the patterns found by SIA are *not* interesting. For example, SIA found around 70000 MTPs in Rachmaninoff's *Prelude* in C♯ minor but a music analyst would probably be interested in only about 100 or so of these. This implies that we need to design heuristics for selecting the interesting patterns in the output of SIA and SIATEC.

It is possible to go some way towards isolating the themes and motives in a piece of music by using just three simple heuristics: *coverage*, *compactness* and *compression ratio*.

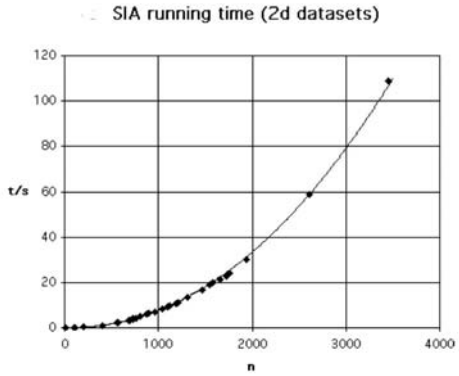

Fig. 14. Experimental running times of SIA on a collection of 2-dimensional datasets. The solid line has the equation $t = Cn^2 \log n$ where C is a suitably chosen constant.

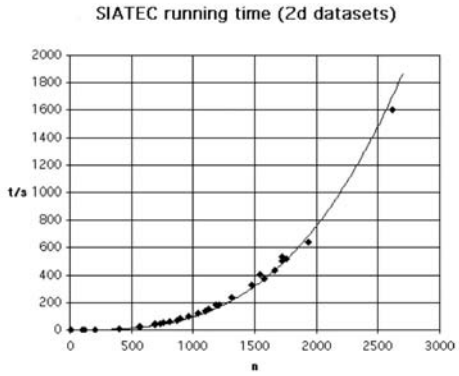

Fig. 15. Experimental running times of SIATEC on a collection of 2-dimensional datasets. The solid line has the equation $t = Cn^3$ where C is a suitably chosen constant.

The coverage of a pattern is the number of distinct points in the dataset in occurrences of the pattern. For example, the coverage of the triangular pattern in Figure 16a is 6 whereas the coverage of the same pattern in Figure 16b is 9. In general, the coverage of a pattern is greater for larger, non-overlapping patterns that occur frequently. If we represent a passage of music as a 2-dimensional dataset like the ones in Figures 8 and 9, then musical themes generally seem to have relatively high coverage.

The compactness of a pattern is the ratio of the number of points in the pattern to the number of points in the region spanned by the pattern. Obviously, the value of compactness depends on how we define "the region spanned by a pattern". For example, we can define it to be the bounding box of the pattern in a pitch-vs.-onset-time representation of the music. If we do this, then the kite-shaped pattern consisting of the round dots in Figure 16d has a compactness of 2/5. However, we could also define the region spanned by a pattern to be the segment of the music containing all the notes whose onset times are greater than or equal to that of the first note in the pattern and less than or equal to that of the last note in the pattern. If we do this, then the compactness of the same kite-shaped pattern in Figure 16 becomes 1/3 (see Figure 16c). Alternatively, we could define the region spanned by a pattern to be the convex hull of the pattern in a pitch-vs.-onset-time representation, in which case, the compactness of the same pattern becomes 2/3 (see Figure 16e). It seems that musical themes generally have at least one occurrence with relatively high compactness. However, other occurrences of the theme may be highly embellished and thus have lower compactness.

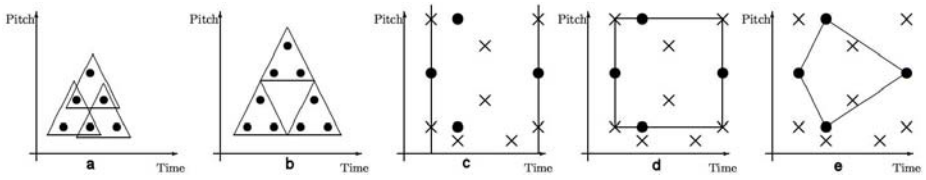

Fig. 16. Examples illustrating coverage, compactness and compression ratio.

Another interesting heuristic, that seems to be useful for isolating themes, is the compression ratio that can be achieved by representing the set of points covered by all the occurrences of a pattern by specifying just one occurrence of the pattern together with all the non-zero vectors by which the pattern is translatable within the dataset. For example, by doing this with the triangular pattern in Figure 16a, we achieve a compression ratio of 6/5. However, in Figure 16b, the same pattern can be used to achieve a compression ratio of 9/5.

8 COSIATEC: Data compression using SIATEC

The heuristics described in the previous section can be used in conjunction with SIATEC to generate a compressed or efficient representation of a dataset. The flow-chart in Figure 17 describes the working of the COSIATEC algorithm. This algorithm takes a multidimensional dataset as input and generates a list of TECs that, taken together, cover the input dataset without any overlapping. Each TEC in the output of COSIATEC is represented as an ordered pair, $\langle P, V \rangle$, where P is a pattern in the TEC and V is the set of non-zero vectors by which P is translatable within the dataset. As shown in Figure 17, COSIATEC first runs SIATEC on the complete dataset, generating a list of TECs in $\langle P, V \rangle$ format. The heuristics described in the previous section are then used to select the "best" of these TECs and this best TEC is printed out in the compact $\langle P, V \rangle$ format. All the points covered by this best TEC are then removed from the dataset. If the dataset is now empty, then the algorithm terminates. If there are still points left in the dataset, the process is repeated with SIATEC being run on the remaining points. Obviously, the degree of compression achieved depends on the amount of repetition in the dataset.

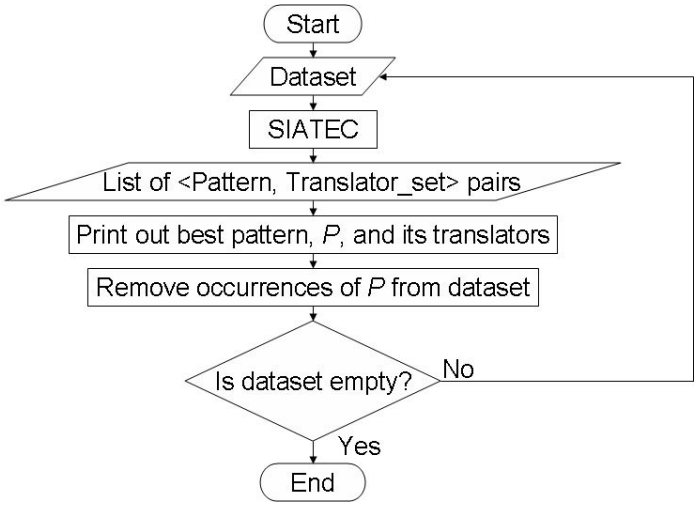

Fig. 17. The COSIATEC data compression algorithm.

9 Using COSIATEC for finding themes and motives in music

The COSIATEC algorithm just described was run on diatonic pitch representations (like the one in Figure 9) of the 15 Two-Part Inventions (BWV 772–786)

by J. S. Bach. The efficient representations generated by COSIATEC for these pieces resembled to an encouraging degree thematic/motivic analyses of the type that one might expect from an expert music analyst. For example, Figure 18 shows the patterns that were generated by COSIATEC on its first iteration for three of the inventions. In each case, the pattern found is a prominent repeated motive in the piece. In particular, the musicologists, Malcolm Boyd [13, p. 96] and Laurence Dreyfus [14, pp. 14–17], both identify the pattern in Figure 18a as being an important motive in this piece. Figure 19 shows the patterns generated by COSIATEC on the second iteration for the same three inventions. Again, the patterns generated are all prominent motives in their respective pieces. For example, Figure 19a is the subject of BWV 772.

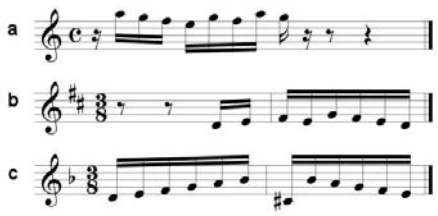

Fig. 18. The patterns generated on the first iteration of COSIATEC for three of J. S. Bach's Two-Part Inventions (a: BWV 772; b: BWV 774; c: BWV 775).

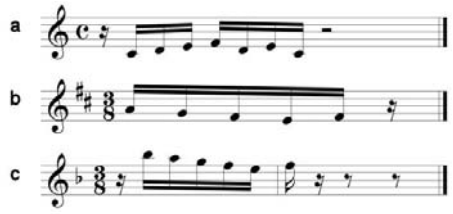

Fig. 19. The patterns generated on the second iteration of COSIATEC for three of J. S. Bach's Two-Part Inventions (a: BWV 772; b: BWV 774; c: BWV 775).

10 SIAM: Finding the maximal matches of a query pattern in a dataset

SIA can easily be adapted for point set pattern matching—or, more specifically, for finding all the maximal matches under translation of a query point set of size m within some dataset of size n. The SIAM algorithm finds, for each possible

vector, the best match for a k-dimensional query pattern in a k-dimensional dataset [15,16,17]. Let's suppose, for example, that we want to find the best match, for each possible vector, of the query pattern in Figure 20 in the dataset in Figure 21. SIAM first sorts the points in the query pattern and the points in the dataset into lexicographical order. Then it computes the vector from each query pattern point to each dataset point, storing these vectors in a vector table as shown in Figure 22. Note that each vector is stored with a pointer to the query point for which it was computed. Then SIAM sorts the vectors in this table to get a list as shown in Figure 23. The maximal match for a given vector is then given by the query points attached to the consecutive occurrences within this list of that vector.

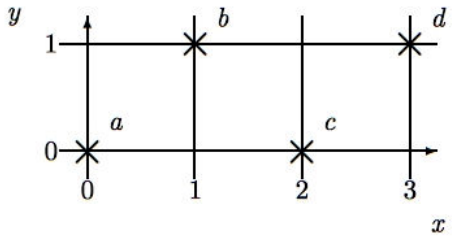

Fig. 20. An example query point set for input to SIAM.

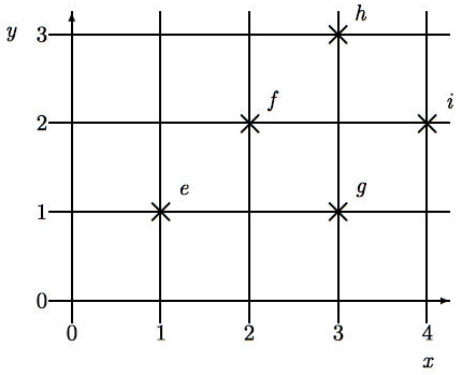

Fig. 21. An example dataset for input to SIAM.

This algorithm runs in $O(knm\log(nm))$ time and $O(knm)$ space in the worst case. However, by storing the origin points in a hash table and hashing the vectors to get the slot indices, the average time complexity can be reduced to $O(knm)$. Note that the naïve method for accomplishing SIAM's task, involving trying all

possible alignments between the query and the dataset and then seeing which query points match up for each alignment, requires $O(knm^2)$ time.

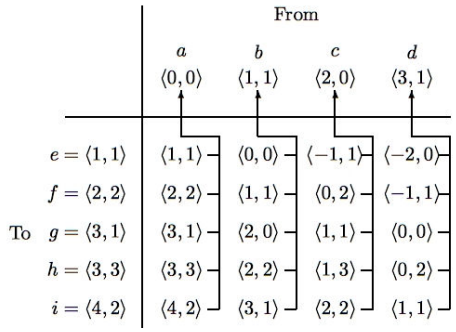

Fig. 22. The vector table generated by SIAM for the query point set in Figure 20 and the dataset in Figure 21.

11 Further developments of SIAM

Ukkonen, Lemström and Mäkinen developed some new algorithms based on SIAM [18]. They present music retrieval algorithms that work not only on point sets but also sets of horizontal line segments in a pitch-time graph (i.e., "piano-roll" representations). Given two point (or line-segment) sets, P and T, of sizes m and n, respectively, they consider the following three problems:

P1. Find all translations of P such that all the onsets in P match with onsets in T.
P2. Find all translations of P such that some of the onsets in P match with onsets in T.
P3. Find translations of P that maximise the overlap between the line segments in P and the line segments in T.

Ukkonen *et al.* [18] provide

1. an algorithm that solves P1 in $O(n)$ average time ($O(mn)$ in the worst case) and $O(m)$ working space;
2. an algorithm that solves P2 in $O(mn \log m)$ time and $O(m)$ space; and
3. an algorithm that solves P3 in $O(n \log n + mn \log m)$ time and $O(n + m)$ space.

All three algorithms are based on a sweepline-like scanning of T [19] and assume that the points (or line segments) in P and T have been sorted into lexicographic order by onset time. P2 is very similar to the problem solved by SIAM but it is not precisely the same, as SIAM finds all the non-empty

VECTOR		DATAPOINT
⟨−2, 0⟩	→	⟨3, 1⟩
⟨−1, 1⟩	→	⟨2, 0⟩
⟨−1, 1⟩	→	⟨3, 1⟩
⟨0, 0⟩	→	⟨1, 1⟩
⟨0, 0⟩	→	⟨3, 1⟩
⟨0, 2⟩	→	⟨2, 0⟩
⟨0, 2⟩	→	⟨3, 1⟩
⟨1, 1⟩	→	⟨0, 0⟩
⟨1, 1⟩	→	⟨1, 1⟩
⟨1, 1⟩	→	⟨2, 0⟩
⟨1, 1⟩	→	⟨3, 1⟩
⟨1, 3⟩	→	⟨2, 0⟩
⟨2, 0⟩	→	⟨1, 1⟩
⟨2, 2⟩	→	⟨0, 0⟩
⟨2, 2⟩	→	⟨1, 1⟩
⟨2, 2⟩	→	⟨2, 0⟩
⟨3, 1⟩	→	⟨0, 0⟩
⟨3, 1⟩	→	⟨1, 1⟩
⟨3, 3⟩	→	⟨0, 0⟩
⟨4, 2⟩	→	⟨0, 0⟩

Fig. 23. The list which results when the vectors in Figure 22 are sorted into lexicographical order. The maximal match for any given vector can be found by reading off the query points attached to consecutive occurrences of that vector in this list.

maximal matches of a query point set P in a text point set T. Nevertheless, the technique used in their solution to P2 for reducing the space complexity from $O(knm)$ to $O(m)$ could also be employed in SIAM. Implementations of the algorithms described by Ukkonen *et al.* [18] are available online at http://www.cs.helsinki.fi/group/cbrahms/demoengine/.

12 Using a fast, randomised version of SIAM for document-level retrieval

Clifford *et al.* [1] have shown that finding the size of the largest maximal match for a query point set in a dataset is 3SUM-hard and therefore unlikely to be solvable in better than quadratic time. However, they have also shown that the problem of multidimensional point-set matching can be reduced to that of 1-dimensional binary wildcard matching. Specifically, they present a new, SIAM-based algorithm, called MSM, in which the data is first randomly projected onto a line in 1 dimension. The length of the data is then reduced using universal hashing. Then the Fast Fourier Transform is used to do binary wildcard matching on the hashed data. Finally, the size of the best match can be checked in $O(m)$ time to find exactly how many points match at the location that can be inferred from that match. The worst-case time complexity of this algorithm is $O(n \log n)$ (i.e., independent of the size of the query point set).

The size of the largest maximal match generated by MSM can be used as a measure of the similarity between any two point sets. It was used in this way on a document-level music information retrieval task in which each of 480 query

documents were searched for in a database containing 2338 documents. Each query document in this test was a representation of a Bach Chorale in which certain notes had been deleted or transposed. All the documents were representations of musical scores, so the onset times and durations in the documents were strictly proportional to their notated values. The performance of the algorithm was compared with that of the OMRAS algorithm [20] and the precision-recall curves for the two algorithms are shown in Figure 24. Note that, on this particular database which consisted only of encodings with no temporal perturbation, the new MSM algorithm proposed by Clifford *et al.* [1] achieved almost perfect precision-recall results.

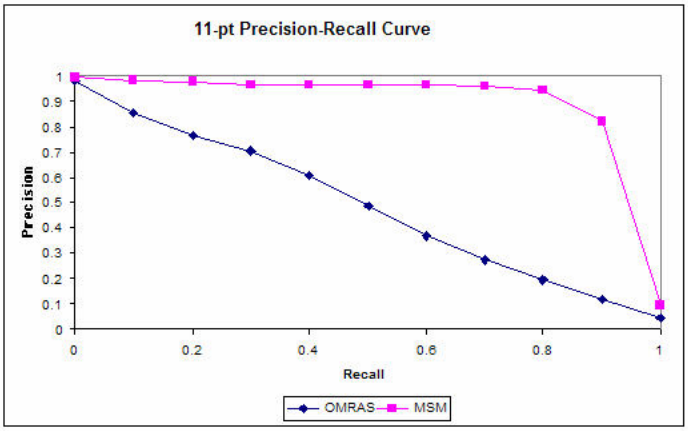

Fig. 24. 11-pt precision-recall curves for OMRAS and MSM when the two algorithms were used to carry out a document-level retrieval task in which each of 480 queries was searched for in a database of 2338 musical score encodings. Each query represented a Bach Chorale in which some of the notes had been deleted or transposed. See [1] for details.

The running time of MSM was then compared with that of the largest common subset algorithm described by Ukkonen *et al.* [18] by running both algorithms on prefixes of various sizes of the first movement of Beethoven's Third Symphony ('Eroica'). The time taken to match each prefix against itself was measured and the results are shown in Figure 25. As can be seen in this figure, MSM was nearly two orders of magnitude faster than Ukkonen *et al.*'s largest common subset algorithm on this task. (Note that the vertical axis has a log scale in this graph.)

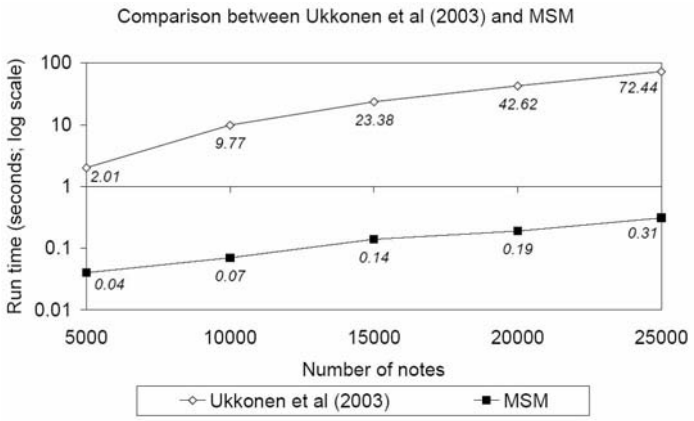

Fig. 25. Results of timing experiment comparing MSM with Ukkonen *et al.*'s [18] largest common subset algorithm. Note that vertical axis has a log scale. See [1] for details.

13 Summary

An algorithm that discovers the significant themes and motives in a musical work could be used in a music information retrieval system for indexing a collection of documents so that it can be searched more rapidly. Moreover, music analysts and psychologists agree that identifying the significant repeated themes and motives in a musical work is an essential step in achieving a rich understanding of it. A musical pattern discovery algorithm can therefore form an essential component in a computational model of music cognition.

Unfortunately, the vast majority of repeated patterns in music are neither intended by the composer nor heard by the listener. An algorithm for discovering the significant repetitions in a musical work must therefore be able to select only those repetitions that would be interesting to an expert listener or music analyst. This problem of selecting only the interesting repeated patterns is made more difficult by the fact that the class of interesting musical repetitions is a very diverse set. There are at least two reasons for this. The first is that the patterns involved in such repetitions vary widely in their structural characteristics. The second is that there are many ways of transforming a musical pattern to obtain a different pattern that is perceived to be a version of the first.

Most previous approaches to pattern matching and pattern discovery in music have been based on the assumption that the musical data will be in the form of strings. However, problems arise when one attempts to use string-based algorithms for finding highly embellished variations on a query pattern or for discovering patterns in polyphonic music in which the voice of each note is unknown.

These problems can be avoided by using multidimensional point sets instead of strings to represent music. In this paper, a number of algorithms have

been described for processing music represented as point sets. Specifically, an $O(kn^2 \log n)$-time algorithm called SIA was described for discovering all the maximal translatable patterns in a k-dimensional point set of size n. Also, an $O(kn^3)$-time algorithm called SIATEC was presented for discovering all the translationally invariant occurrences of all the maximal translatable patterns in a k-dimensional point set of size n.

Unfortunately, many of the patterns found when SIA and SIATEC are run on music data are not interesting. Three heuristics (coverage, compactness and compression ratio) were therefore proposed for identifying those patterns in the output of SIA and SIATEC that correspond to themes, motives and other memorable musical patterns.

These heuristics were used in conjunction with SIATEC to construct a data compression algorithm called COSIATEC. This algorithm generates an efficient representation of a point set in the form of a list of TECs, each TEC being represented as an ordered pair, $\langle P, V \rangle$, in which P is one pattern in the TEC and V is the set of vectors by which P is translatable in the dataset. When COSIATEC was run on the 15 Two-Part Inventions by J. S. Bach, it was found that it often generated prominent motives and themes on its first and second iterations.

I then briefly described a pattern matching algorithm based on SIA called SIAM. SIAM is an $O(kmn \log(mn))$-time algorithm for finding, for every vector, the maximal match for a k-dimensional query point set in a k-dimensional dataset.

I then reviewed some recent work by Ukkonen *et al.* [18] which showed how SIAM can be generalised to work not only on point sets but also "piano-roll" music representations. Ukkonen *et al.* also showed how the space complexity of SIAM can be reduced to $O(km)$. Finally, I reviewed some recent work by Clifford *et al.* [1] which proves that the problem solved by SIAM is not solvable in subquadratic time. However, Clifford *et al.* also show how, by using randomisation, the problem of multidimensional point-set matching can be reduced to that of 1-dimensional binary wildcard matching. Clifford *et al.* use this to develop an $O(n \log n)$ algorithm, called MSM, for finding the size of the largest maximal subset match under translation between two point sets. This algorithm was used in a document-level music retrieval task to measure the similarity between a query document and each document in a database. When the documents were score encodings without temporal perturbation, MSM produced almost perfect precision-recall results and was nearly two orders of magnitude faster than Ukkonen *et al.*'s [18] largest common subset algorithm.

14 Future work

The SIA-based algorithms described in this paper should be compared rigorously with methods that have been developed in more mature fields such as computer vision, computational geometry and graph matching.

Given the success of the MSM algorithm, an obvious next step would be to attempt to improve the time complexity of SIA and SIATEC by using similar randomisation techniques.

All the algorithms described in this paper work well on musical score representations in which the onset times and durations are strictly proportional to their notated values. However, they tend to fail when there are tempo changes or local temporal perturbations in the data such as one will tend to find in data derived from human performances. Versions of the algorithms described in this paper therefore need to be developed that can handle such tempo fluctuations and perturbations.

Acknowledgements

SIA, SIATEC, SIAM and COSIATEC were developed in collaboration with Geraint A. Wiggins and Kjell Lemström during a project funded by EPSRC grant number GR/N08049/01. The other work described in this paper was completed during a project funded by EPSRC grant number GR/S17253/02. I thank Tim Crawford and Remco Veltkamp for inviting me to attend the Dagstuhl Seminar on Content-Based Retrieval (06171) held at Schloss Dagstuhl from the 23rd to the 28th of April 2006.

References

1. Clifford, R., Christodoulakis, M., Crawford, T., Meredith, D., Wiggins, G.: A fast, randomised, maximum subset matching algorithm for document-level music retrieval. In: Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR 2006) (September 8–12), Victoria, Canada (2006)
2. Lerdahl, F., Jackendoff, R.: A Generative Theory of Tonal Music. MIT Press, Cambridge, MA (1983)
3. Schenker, H.: Harmony. University of Chicago Press, London (1954) Edited by Oswald Jonas and translated by Elisabeth Mann Borgese from the 1906 German edition.
4. Bent, I., Drabkin, W.: Analysis. New Grove Handbooks in Music. Macmillan (1987)
5. Lemström, K.: String Matching Techniques for Music Retrieval. PhD thesis, University of Helsinki, Faculty of Science, Department of Computer Science (2000) Report A-2000-4.
6. Hsu, J.L., Liu, C.C., Chen, A.L.: Efficient repeating pattern finding in music databases. In: Proceedings of the 1998 ACM 7th International Conference on Information and Knowledge Management, Association of Computing Machinery (1998) 281–288
7. Rolland, P.Y.: Discovering patterns in musical sequences. Journal of New Music Research **28** (1999) 334–350
8. Mongeau, M., Sankoff, D.: Comparison of musical sequences. Computers and the Humanities **24** (1990) 161–175

9. Cambouropoulos, E., Crochemore, M., Iliopoulos, C.S., Mouchard, L., Pinzon, Y.J.: Computing approximate repetitions in musical sequences. International Journal of Computer Mathematics **79** (2002) 1135–1148
10. Meredith, D.: The *ps13* pitch spelling algorithm. Journal of New Music Research **35** (2006) In production. Draft available online at <http://www.titanmusic.com/papers/public/meredith-ps13-jnmr.pdf>.
11. Meredith, D.: Computing Pitch Names in Tonal Music: A Comparative Analysis of Pitch Spelling Algorithms (2006) D.Phil. dissertation, Faculty of Music, University of Oxford. Draft available online at <http://www.titanmusic.com/papers/private/meredith-dphil.pdf>. To obtain username and password, send request to <dave@titanmusic.com>.
12. Meredith, D., Lemström, K., Wiggins, G.A.: Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. Journal of New Music Research **31** (2002) 321–345 Available online at <http://taylorandfrancis.metapress.com/link.asp?id=yql23xw0177lt4jd>.
13. Boyd, M.: Bach. The Master Musicians. J. M. Dent, London (1983)
14. Dreyfus, L.: Bach and the Patterns of Invention. Harvard University Press, Cambridge, Mass. (1996)
15. Meredith, D., Lemström, K., Wiggins, G.A.: Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. In: Cambridge Music Processing Colloquium (28 March 2003), Cambridge University Engineering Department (2003) Available online at <http://www.titanmusic.com/papers/public/cmpc2003.pdf>.
16. Wiggins, G.A., Lemström, K., Meredith, D.: SIA(M)ESE: An algorithm for transposition invariant, polyphonic, content-based music retrieval. In: 3rd International Symposium on Music Information Retrieval (ISMIR 2002), 13–17 September 2002, IRCAM, Centre Pompidou, Paris, France. (2002) 283–284 Available online at <http://ismir2002.ismir.net/proceedings/03-SP03-4.pdf>.
17. Meredith, D., Wiggins, G.A., Lemström, K.: Pattern induction and matching in polyphonic music and other multidimensional datasets. In Callaos, N., Zong, X., Vergez, C., Pelaez, J.R., eds.: Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI2001), July 22-25. Volume X., Orlando, FL. (2001) 61–66 Paper and slides available online at <http://www.titanmusic.com/papers.html>.
18. Ukkonen, E., Lemström, K., Mäkinen, V.: Geometric algorithms for transposition invariant content-based music retrieval. In: Proceedings of the Fourth International Conference on Music Information Retrieval, Baltimore (ISMIR 2003, 26–30 October 2003). (2003) 237–238 Available online at <http://ismir2003.ismir.net/papers/Ukkonen.PDF>.
19. Bentley, J., Ottmann, T.: Algorithms for reporting and counting geometric intersections. IEEE Transactions on Computers **C** (1979) 643–647
20. Pickens, J., Bello, J.P., Monti, G., Sandler, M., Crawford, T., Dovey, M., Byrd, D.: Polyphonic score retrieval using polyphonic audio queries: A harmonic modeling approach. Journal of New Music Research **32** (2003) 223–236 Available online at <http://taylorandfrancis.metapress.com/link.asp?id=3tlbdymc8qjj5cy2>.