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

David Meredith
Department of Computing, City University, London,
Northampton Square, London, EC1V 0HB, United Kingdom.
Tel./Fax: +44 (0)1945 870999.
`dave@titanmusic.com` `http://www.titanmusic.com`

Kjell Lemström
University of Helsinki, Department of Computer Science,
P.O.Box 26 (Teollisuuskatu 23), FIN-00014 Univ. of Helsinki, Finland.
Tel.: +358 9 191 44209. Fax: +358 9 191 44441.
`klemstro@cs.helsinki.fi` `http://www.cs.Helsinki.fi/u/klemstro`

Geraint A. Wiggins
Department of Computing, City University, London,
Northampton Square, London, EC1V 0HB, United Kingdom.
Tel.: +44 (0)20 7040 8848. Fax: +44 (0)20 7040 8587.
`geraint@soi.city.ac.uk` `http://www.soi.city.ac.uk/~geraint/`

Address for correspondence:
David Meredith, The Barn, Middle Broad Drove, Tydd St. Giles, Wisbech,
Cambs., PE13 5PA, UK. Tel./Fax: +44 (0)1945 870999. `dave@titanmusic.com`

Running head title: Discovering Repeated Patterns in Music

May 9, 2002

[*]The algorithms described in this paper are the subject of an active patent application submitted on 23 May 2001 (Meredith *et al.*, 2001).

Abstract

In previous approaches to repetition discovery in music, the music to be analysed has been represented using strings. However, there are certain types of interesting musical repetitions that cannot be discovered using string algorithms. We propose a *geometric* approach to repetition discovery in which the music is represented as a *multidimensional dataset*. Certain types of interesting musical repetition that cannot be found using string algorithms can efficiently be found using algorithms that process multidimensional datasets. Our approach allows polyphonic music to be analysed as efficiently as monophonic music and it can be used to discover polyphonic repeated patterns 'with gaps' in the timbre, dynamic and rhythmic structure of a passage as well as its pitch structure. We present two new algorithms: `SIA` and `SIATEC`. `SIA` computes all the maximal repeated patterns in a multidimensional dataset and `SIATEC` computes all the occurrences of all the maximal repeated patterns in a dataset. For a k-dimensional dataset of size n, the worst-case running time of `SIA` is $O(kn^2 \log_2 n)$ and the worst-case running time of `SIATEC` is $O(kn^3)$.

1 Repetition in music

In this paper we address the problem of designing an algorithm for discovering repetitions in music. Such an algorithm would have both scientific and engineering applications. For example, it could form an important component in a computational model of expert music cognition and it could also be used to build useful software tools for music analysts and composers. A music analyst might find such an algorithm useful for discovering characteristic structural features in the works of a particular composer or for analysing the structure of a work into its elemental 'building blocks'. A composer who is suffering from writer's block could use such an algorithm to find significant repeated structures in an unfinished work thus discovering a fresh perspective that might stimulate further progress on the work. Another important practical use for such an algorithm is for music database indexing (Lemström *et al.*, 1998; Lincoln, 1967).

Many music analysts and music psychologists (see, for example, Bent and Drabkin, 1987; Lerdahl and Jackendoff, 1983; Nattiez, 1975; Ruwet, 1972; Schenker, 1954) have stressed that the identification of perceptually significant repetitions is an essential step in the process by which an expert listener interprets a musical work. Heinrich Schenker, for example, claimed that repetition 'is the basis of music as an art' (Schenker, 1954, p. 5). Bent and Drabkin (1987, p. 5) observed that 'the central analytical act' in all forms of music analysis is 'the test for identity' and Lerdahl and Jackendoff 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.
>
> (Lerdahl and Jackendoff, 1983, p. 52)

However, the vast majority of exact repetitions in music are *not* perceptually significant (see section 8 below). Therefore, the task of developing an algorithm that isolates perceptually significant repetitions in music involves formally characterising what it is about these interesting repetitions that distinguishes them from the many repetitions that the listener does not notice and the analyst does not consider to be important.

Unfortunately, the class of perceptually significant repetitions in music 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. Second, there are many different ways in which a pattern can be modified to give a second pattern that is perceived to be a version of it.

A perceptually significant repeated pattern may be a small motif consisting of just a few notes (see Figure 1) or it might be a whole section of a work (e.g., the exposition of a sonata-form movement). In voiced music, a repeated pattern may only contain notes from one voice (e.g., a fugal subject) or it may contain notes from two or more voices. The occurrences of a pattern may overlap in time as they do, for example, in *stretto* passages in baroque contrapuntal works; or they may occur consecutively, as often happens, for example, in Debussy's music (Ruwet, 1972, pp. 70–99); or they may be widely separated in time as they are in the case of the exposition and recapitulation of a sonata-form movement.

PLEASE INSERT FIGURE 1 ABOUT HERE.

A significant repeated pattern may be *temporally compact*—that is, it may contain all the notes in the music that occur within the time interval spanned by the pattern. On the other hand, in a fugue, each statement of the subject only contains all the notes *in a single voice* that occur within the time interval that it spans.

In baroque and renaissance music, it was common for singers and instrumentalists to embellish their parts by replacing a long note with a sequence of shorter notes. This technique is called *diminution* (Forte and Gilbert, 1982, Chapter 1). For example, in Figure 2, pattern B is a diminution of pattern A: notes 1, 5, 9 and 13 in B form a modifed restatement of A. This suggests that a pattern involved in a perceptually significant repetition may not even contain all the notes in a single voice that occur within the time interval spanned by the pattern.

PLEASE INSERT FIGURE 2 ABOUT HERE.

The term *diminution* is also (confusingly) used to signify a contrapuntal technique in which the durations of the notes in a theme are divided by some constant value. Bach uses this technique in *Contrapunctus VI* from *Die Kunst der Fuge* (Figure 3) where the first alto entry of the subject is obtained from the opening bass entry by halving the note durations. Bach uses the inverse of diminution—*augmentation*—in *Contrapunctus VII* of *Die Kunst der Fuge* where the first bass entry is obtained from the alto entry by *multiplying* the note durations by 4. If a piece is represented as a set of points in a two-dimensional space where the dimensions are pitch and time and each point represents a note (see section 4 below), then the musical transformations of augmentation and diminution correspond to the geometrical transformation of *central dilatation* (see Borowski and Borwein, 1989, p. 162, s.v. **dilatation**).

PLEASE INSERT FIGURE 3 ABOUT HERE.

Figure 3 also illustrates the contrapuntal technique of *inversion*: the first alto entry of the subject in Figure 3 is an inversion of the soprano entry. If a musical passage is represented as a set of points in a two-dimensional space where the dimensions represent time and pitch, then musical inversion corresponds to geometrical reflection about an axis parallel to the time axis. The techniques of augmentation/diminution and inversion were sometimes used in combination in baroque music. For example, the soprano entry in Figure 3 is an inverted diminution of the bass entry.

It is also possible to find perceptually significant repetitions in which one pattern involved in the repetition is the reverse (or retrograde) of the other. This occurs, for example, in the 'Canon cancrizans' from Bach's *Musikalisches Opfer*. If we represent a musical passage as a set of points in a two-dimensional space in which the dimensions represent pitch and time, then the musical transformation of reversal corresponds to a geometrical transformation of reflection about an axis parallel to the pitch axis.

There are many other ways in which a musical pattern may be modified to give a second pattern that is perceived to be a version of it but even from the foregoing discussion it should be clear that the class of perceptually significant musical repetitions is a very diverse set. We have also shown that if the music to be analysed is appropriately represented as a set of points in a multidimensinal space, then certain types of thematic modification (e.g., diminution, augmentation, inversion, reversal and transposition) correspond directly to geometrical transformations within such a multidimensional representation.

2 Representing music using strings

In section 6 we present two new algorithms, `SIA` and `SIATEC`, that can be used for discovering types of musical repetition that cannot be discovered using other existing algorithms. In previous approaches to repetition discovery in music, the music to be analysed has been represented using strings. There are many different meaningful and potentially useful ways of representing a piece of music as a string (or as a set of strings). However, each of the string-based music representations that have been proposed in the literature falls into one of the following two categories:

1. *Event strings*: Each symbol in the string $x = x_1 \dots x_n$ represents a musical event. In this case, if x_i and x_j are symbols in x, then $j > i$ if and only if the event represented by x_j occurs later in the music than x_i.

2. *Interval strings*: Each symbol x_k represents the transformation required to transform an event $e_1(x_k)$ into an event $e_2(x_k)$ that follows it in the music. If x_i, x_j are any two symbols in such a string x, then $j > i$ if and only if $e_1(x_i) \prec e_2(x_i) \preceq e_1(x_j) \prec e_2(x_j)$ where $\prec$ means 'precedes' and $\preceq$ means 'precedes or is equal to'.

Perhaps the most obvious example of an event string representation is one in which each symbol represents the pitch of a note event.[1] A rhythm can be encoded using an event string consisting simply of a sequence of durations. A monophonic melody can be represented as a string in which each symbol represents both the pitch and duration of a single note. A set of such event strings could be used to represent a polyphonic piece, each string representing a different voice. A passage of homorhythmic music can be represented by a single event string in which each symbol represents a single chord. A polyphonic passage can also crudely be represented as a single event string in which each symbol represents the set of pitches starting at a given instant in the music (Dovey, 1999; Dovey and Crawford, 1999, 2000; Lemström, 2000). This method of representation often suffices for music information retrieval purposes.

Conklin and Anagnostopoulou (2001) describe a pattern discovery technique based on multiple viewpoints (Conklin and Witten, 1995). In this approach, a number of different strings are used to represent any given passage of music, each string representing a particular viewpoint. For example, one of Conklin and Anagnostopoulou's representations describes melodic contour, another represents pitch, a third represents the inter-onset intervals between consecutive events and so on. Conklin and Anagnostopoulou also use event strings to represent, in a rudimentary way, musical structure on hierarchical levels higher than the note level. For example, in one of their representations, only the first note in each crotchet beat is represented; and in another, only the first note in each *bar* is represented.

The most obvious example of an interval string representation is where each symbol represents the pitch interval between two consecutive notes in a single voice. It is usually (but not always—see Lemström and Ukkonen, 2000) easier to find transposition-invariant occurrences of a pattern in such a representation than it is in an event string in which each symbol represents the pitch of a note (Lemström, 2000, p. 24). Such a representation may be augmented by adding duration information and a polyphonic piece could be encoded as a set of such interval strings. Indeed, the pitch interval and rational duration information of a note may be encoded using a single value allowing both transposition and tempo invariances to be obtained using linear strings (Lemström *et al.*, 1999). Lemström and Tarhio (2000) describe how a piece of polyphonic music may be represented as a string of bitvectors, each bitvector representing the intervals between the pitches that occur at an instant and the pitches that occur at the next instant at which a new pitch is sounded. There are many other interval systems on which one could base an interval string music representation. Lewin (1987, Chapter 2) gives a number of examples of *Generalized Interval Systems* (*GIS*s) on which such representations could be based. There is thus a multiplicity of ways in which one can represent musical passages using event strings and interval strings.

[1]The pitch of the note may be represented in various ways. For example, one may use diatonic or chromatic pitch, MIDI number, pitch class, or some more complex pitch representation such as Cambouropoulos's (1996) *GPIR* or one of Meredith's (1999; 2001) or Brinkman's (1986; 1990) representations.

3 A survey of algorithms for discovering repetitions in music

One component of Cambouropoulos's (1998) *General Computational Theory of Musical Structure* is a 'Sequential Pattern Induction Algorithm' (*SPIA*) that identifies repetitions in a musical surface. Cambouropoulos (1998, pp. 88–90) claims that his *SPIA* can profitably be used on various different event string and interval string representations. The original version of Cambouropoulos's *SPIA* was an implementation of a pattern induction algorithm described by Crow and Smith (1992, pp. 46–52). This algorithm computes all the maximal repeated factors in a string: if $x = x_1 \ldots x_n$ is a string then w is a *factor* of x if and only if there exist i and j such that $w = x_i \ldots x_j$. In other words, a factor is a pattern that does not contain any 'gaps'. In a later version of his *SPIA*, Cambouropoulos replaced the Crow-Smith algorithm with a more efficient algorithm based on a partitioning technique for refinements of equivalence relations described by Cardon and Crochemore (1982) and Crochemore (1981). The algorithm is given in full and analysed by Iliopoulos and Mouchard (1999, pp. 265–267). This algorithm finds all occurrences of all the distinct factors in a string in a worst-case running time of $O(n \log n)$ for a string of length n.

This partitioning algorithm and Crow and Smith's (1992) algorithm are two examples of string-processing algorithms that can be used for discovering various classes of repeated factors in strings. Other algorithms of this type include Crochemore's (1981) $O(n \log n)$–time algorithms for finding the maximal periodic factors in a string: a string w is a *periodic factor* of another string x if and only if w is a factor of x and w can be formed by concatenating two or more copies of some string y. Apostolico and Preparata (1983) give another algorithm that computes the same as Crochemore's partitioning algorithm but uses more complex data structures. Apostolico and Ehrenfeucht (1993) described an algorithm based on that of Apostolico and Preparata (1983) for computing all the maximal quasiperiodicities in a string in time $O(n \log^2 n)$: a string w is a *quasiperiodicity* in another string x if and only if w is a factor of x and there exists another string $y \neq w$ such that every position of w falls within some occurrence of y in w. Iliopoulos and Mouchard (1999) give an $O(n \log n)$ algorithm that computes all the maximal quasiperiodicities in a string. Their algorithm shadows Crochemore's (1981) algorithm. It is faster by a factor of $O(\log n)$ than the algorithm described by Apostolico and Ehrenfeucht (1993) for solving the same problem. Iliopoulos and Mouchard (1999, p. 263) claim that the computation of maximal quasiperiodicities 'has direct application in music analysis'. For example, they suggest that one might want to find all repeated factors in a piece that are at most k positions apart or find all repeated non-overlapping factors of length greater than k for some given integer k.

A factor w of a string x is called a *cover* of x if and only if every position in x falls within an occurrence of w in x. For example, *abca* is a cover of *abcabcaabca*. The *shortest-cover problem* (also known as the *superprimitivity test*) is that of computing the shortest cover of a string. The *all-covers problem* is that of computing all the covers of a string. Apostolico *et al.* (1991) and Breslauer (1992) have presented linear-time algorithms for solving the shortest-cover problem. Breslauer (1994) also presented a parallel algorithm that computes the shortest-cover of a string in $O(\log \log n)$ time using an $\left(\frac{n \log n}{\log \log n}\right)$–processor concurrent-read-concurrent-write parallel random access machine (CRCW PRAM). Iliopoulos and Park presented optimal $O(\log \log n)$–time (thus work-time optimal) parallel algorithms for the shortest-cover problem (Iliopoulos and Park, 1994) and the all-covers problem (Iliopoulos and Park, 1996). Moore and Smyth (1994) presented a linear-time algorithm for the all-covers problem and Li and Smyth (1999) gave a linear-time on-line algorithm for computing all the covers of all prefixes of a string. Iliopoulos *et al.* (1993) introduced the idea of a seed: a factor w of a string x is called a *seed* of x if x is a factor of a string z which is covered by w. For example, $w = abca$ is a seed of $x = abcabcaabc$ because x is a factor of $z = abcabcaabca$ and w covers z. Iliopoulos *et al.* (1993) gave an $O(n \log n)$–time algorithm for computing all the seeds in a string. A parallel algorithm for solving the same problem was presented by Ben-Amram *et al.* (1994) that requires $O(\log n)$ time and $O(n \log n)$ work.

Hsu *et al.* (1998) used a dynamic programming technique to find repeating factors in strings representing monophonic melodies. Though having an $O(n^4)$ worst case time complexity, it works much more efficiently in practice. In their study, Hsu *et al.* did not consider transposition invariance, but this can be obtained straightforwardly by using an interval string representation instead of an event string representation. First they use a correlative matrix, whose processing takes $O(n^2)$ time. The output of this phase is a candidate set containing all the factors in the input string (even those that appear only once), together with their frequency. In a second phase, each candidate which is a factor of another candidate c (and whose frequency does not exceed the frequency of c) is removed from the candidate set. In a pathological case the number of required operations for this phase can be $O(n^4)$. However, there are rather fewer candidates to be considered in practice.

These and other algorithms for finding repeated factors in strings can be used to find a restricted (but nonetheless important) class of repetitions in music. For example, if a fugal work is represented as a set of pitch interval strings, each string representing a voice and each symbol in each string representing the diatonic interval between two consecutive notes in a voice, then all the occurrences of the fugal subject could be discovered using an algorithm that only found the exactly repeated factors in a string. The repetition in Figure 2 could be discovered using an algorithm for discovering exactly repeated factors if one were to adopt one of Conklin and Anagnostopoulou's (2001) suggestions and represent the music as a string in which each symbol represents the pitch of the first note in a crotchet beat.

In baroque contrapuntal keyboard works, the voice structure of the music is often very clearly indicated in the score by cues such as the directions of note stems (see, for example, Figure 3). However, in nineteenth and twentieth century piano music, the voicing is often ambiguous and not explicitly represented in the score. If one wants a 'neutral' representation of a score that does not represent just one particular encoder's *interpretation* of the score, then one should omit information that is not explicitly represented in the score. A neutral representation of a keyboard work will therefore often contain no information about voicing. A symbolic representation of what a listener hears when he or she listens to a performance of a work for piano or harpsichord must also not contain any explicit voicing information. Such a representation might be used, for example, as input to a repetition-discovery algorithm forming one part of a computational model of expert listening. Many music databases contain music in the form of MIDI files but a MIDI representation of a polyphonic keyboard work is often structured so that all the notes are in a single channel. Such a file therefore contains no voicing information. In such situations it might be useful to have an algorithm that can discover repetitions in music representations that contain no information about voicing.

Figure 4 shows one natural way of partitioning the notes in the passage in Figure 1 into monophonic voices. The music in Figure 4 could be represented using four pitch interval strings, each string representing the sequence of pitch intervals in one of the four monophonic parts shown. If the music were represented in this way, then a string processing algorithm for finding exactly repeated factors might discover the ten statements of pattern B1 shown in Figure 4 but no algorithm for finding exactly repeated factors would discover the five occurrences (A1–A5) of the rising bass pattern in Figure 1. If we want to use an algorithm that only finds exactly repeated factors for discovering the set of patterns A1–A5 in Figure 1, then we have to allow the 'voices' or 'streams' into which we partition the music to be polyphonic. Figure 5 shows a natural way of partitioning the notes in the passage into three parts, one of which is polyphonic. If each of the three polyphonic 'streams' in Figure 5 is represented by an interval string in which each symbol represents the transformation required to convert a chord into the next chord that occurs in the part then the five occurrences of the boxed pattern would be represented by identical factors in the interval string representation of the part written on the lowest staff in Figure 5.

PLEASE INSERT FIGURE 4 ABOUT HERE.

PLEASE INSERT FIGURE 5 ABOUT HERE.

However, before analysing a work, the user has no idea which particular ways of partitioning it into polyphonic 'streams' will result in the discovery of perceptually significant repetitions. To be sure of discovering all such repetitions, the user would therefore have to consider all possible ways of partitioning the music into polyphonic streams which would require 2^n representations for a piece with n voices. This would be completely impractical for any work with more than, say, four or five parts.

The foregoing discussion demonstrates that the class of perceptually significant musical repetitions that can be discovered using an algorithm that computes the exactly repeated factors in a string is heavily dependent upon the particular type of string representation used. To discover all those repetitions in which the repeated patterns can be represented as identical factors in a string, one has to use not only a variety of algorithms but also a multitude of distinct representations.

So far we have only discussed algorithms that discover *exactly* repeated factors in strings. There also exist algorithms for discovering various types of *approximately* repeated patterns. For example, Cambouropoulos *et al.* (1999, pp. 137–140) give two $O(n^2)$–time algorithms for computing certain types of approximately repeating patterns in a string.

Two strings are said to be *k-approximately similar* if one can be obtained from the other by using k or fewer *editing operations* (e.g., deletion, insertion, substitution) (Crochemore and Rytter, 1994).

The minimum number of edit operations required to transform one string into another is called the *edit distance* between the two strings.

In order to be able to measure the overall similarity between two strings representing (monophonic) musical passages, Mongeau and Sankoff (1990) modified the well-known dynamic programming approach for calculating the edit distance between two strings (Ukkonen, 1985). In addition to the traditional tool-box of editing operations (containing insertion, deletion, and substitution) they introduced two novel musically motivated operations called *fragmentation* and *consolidation*. By limiting the number of notes that can be involved in a fragmentation or consolidation with suitably selected constants, the original time complexity of the dynamic programming approach (i.e., $O(mn)$, m and n denoting the lengths of the two strings being compared) can be preserved. They also showed that the problem of finding *local similarities* in any two monophonic musical passages given to the algorithm as arguments can be solved by a straightforward modification of their method. Thus, by making both arguments of the algorithm the same, all the k–approximately similar repeated patterns can be discovered.

Another approach to discovering approximately repeated patterns in music is presented by Rolland (1999). Like Mongeau and Sankoff (1990), Rolland used an extended toolbox of editing operations. In his method, the length of the patterns involved in the discovered repetitions is limited by setting a minimal and maximal length for them.[2] Then, all the patterns falling in the allowed range are extracted from the musical string. Having created a node for a pattern, it is compared against other patterns (whose length is close enough to that of the considered pattern). If their similarity exceeds a given threshold value, an arc between the two nodes corresponding to these patterns is created and the weight of this arc is set according to their similarity. In this way, the related patterns form *stars* in a similarity graph. Finally, the patterns are sorted by their prominence: for every pattern in the similarity graph, the weights of the arcs by which they are connected to other nodes are accumulated. Then, the patterns are listed in descending order according to the accumulations. Rolland claims an overall time complexity of $O(n^2)$. However, if the algorithm is required to find patterns of any size, this time complexity rises to (at least) $O(n^4)$, making it less efficient in the worst case than `SIATEC`. Also, unlike `SIATEC` and `SIA`, Rolland's algorithm is currently limited to the discovery of patterns in monophonic sources.

It was mentioned above (p. 7) that an algorithm for finding exactly repeated factors could be used to find the repetition in Figure 2 but only if the music were represented as a string in which only the first note in each crotchet beat were represented. However, such an *ad hoc* expedient would not suffice for detecting the fact that pattern B in Figure 6 is an embellished restatement of pattern A. This is because the notes in pattern A do not all fall on the beat. It might be suggested that a repetition like this could be discovered using an algorithm such as Rolland's that discovers k–approximately similar patterns.

PLEASE INSERT FIGURE 6 ABOUT HERE.

[2] In `SIA` and `SIATEC` there are no such bounds—the patterns discovered may be of any size.

Recall that such an algorithm judges two patterns to be 'similar' if and only if the number of edit operations required to transform one into the other is less than some threshold k. However, for the two patterns in Figure 6 to be judged 'similar' by such an algorithm, this threshold k would have to be set to at least 14 to allow for the 14 insertions required to transform pattern A into B. But this threshold is far too high in general, because, with such a high threshold, the algorithm would judge many highly dissimilar patterns to be 'similar'. An algorithm for discovering k–approximately similar patterns therefore cannot be used to find repetitions in which one pattern is a highly embellished version of the other. However, such an algorithm *can* be used to find repetitions in which the two patterns differ by only one or two notes.

So far, we have only considered algorithms that discover repeated factors and repetitions in which the patterns are k-approximately similar. We now briefly review algorithms that discover repeated *subsequences*—that is, patterns with an arbitrary number of 'gaps'. Formally, a string w is a *subsequence* of another string x if and only if x can be transformed into w by deleting zero or more symbols in x. The problem of discovering repeated patterns within a string is closely related to the following well-known string-processing problem: given a finite set W of words, find a pattern, p, that is the longest *substructure* (i.e., factor or subsequence) of every word in W. Note, that W may have been obtained from a long string S that has been divided into $|W|$ factors. It is known (Crochemore and Rytter, 1994, p. 25), that if $|W|$ is not constant and the substructures to be considered are subsequences, the problem becomes NP-complete. However, if factors instead of subsequences are considered, the problem is solvable in polynomial time. This illustrates that finding repetitions in which the patterns can have an arbitrary number of 'gaps' (i.e., repeated subsequences) is much more complex than finding repetitions without gaps (i.e., repeated factors).

Very little effort has been made so far to solve the problem of discovering repeated subsequences in strings. However, an algorithm for discovering subsequences might be able to discover repetitions such as the one shown in Figure 6 without also computing many spurious instances of repetition. This is closely related to the bioinformatics problem of DNA and protein sequence alignment. However, in some respects, the discovery of repeated subsequences in DNA and proteins is simpler than the discovery of musically significant patterns in polyphonic music. This is because DNA and proteins can, at least at the primary level of structure, be appropriately modelled as 1-dimensional strings of symbols taken from a highly restricted alphabet; whereas much polyphonic music cannot even be appropriately represented as a *set* of 1-dimensional strings and the musical 'alphabets' used are (at least in principle if not generally in practice) infinite and multidimensional.

Nevertheless, we shall outline, as an example, a string matching approach to the discovery of repeated subsequences in 1-dimensional strings given by Floratos and Rigoutsos (2000). The idea is to start with initial patterns of a given length (possibly containing also 'don't care' characters), and proceed recursively to generate longer and longer patterns appearing in the data set. Floratos and Rigoutsos attempt to avoid the inherent NP-hardness of the problem by limiting the length of the considered patterns. With the aid of two suffix structures, they attempt to extend the considered patterns both backwards (by assigning a prefix to the pattern) and forwards (by assigning a suffix).

Finally, the generated patterns representing exactly the same subsequences in a different way, are combined in a *maximal pattern*.

4 Representing music using multidimensional datasets

Previous approaches to repetition-discovery in music have been based on the assumption that the music to be analysed is represented using strings. As shown above, there exist string-processing algorithms for discovering repeated factors as well as various types of approximately repeated patterns and subsequences. However, to find all the interesting repetitions in a musical passage using a string-based approach, one has to run various algorithms on a multitude of different representations of the passage. Moreover, when the music to be analysed is polyphonic, there are certain types of repetition that cannot be discovered if the music is represented using strings.

We have avoided these problems by developing a new approach in which the music to be analysed is represented as a *multidimensional dataset*. This allows many different types of interesting repetition to be discovered efficiently by running appropriately designed algorithms (such as `SIA` and `SIATEC`) on just a small number of orthogonal projections of a single multidimensional representation of the music. Repetitions such as the one in Figure 6 can easily be found using this approach as can repeated polyphonic patterns in unvoiced polyphonic music such as keyboard music (i.e., 'unstructured repetitions'—see Crawford *et al.*, 1998, p. 88). The so-called 'distributed matching' problem (Crawford *et al.*, 1998, p. 84) can also be solved using this new geometric approach. By representing music using multidimensional datasets we can dispense with the need for using a multitude of different representations, analyse complex polyphonic music as efficiently as monophonic music and discover repetitions in the timbre, dynamic and rhythmic structure of a passage as well as its pitch structure. We can also discover repeated patterns that would, in a string-based approach, be represented as subsequences—i.e., 'patterns with gaps'.

Informally, a multidimensional dataset can be visualized as a set of points in a multidimensional space. Before giving a formal definition, we need to define some basic concepts. A *vector* is a k-tuple of real numbers viewed as a member of a k-dimensional Euclidean space (Borowski and Borwein, 1989, p. 624, s.v. **vector**, sense 2). A vector in a k-dimensional Euclidean space will be represented here as an ordered set of k real numbers. An object is a *k-dimensional vector* if and only if it is a k-tuple of real numbers (i.e., an ordered set of real numbers with cardinality k). An object is a *vector set* if and only if it is a set of vectors. An object is a *k-dimensional vector set* if and only if it is a vector set in which every vector has cardinality k. An object may be called a *pattern* or a *dataset* if and only if it is a k-dimensional vector set. An object may be called a *datapoint* if and only if it is a vector in a pattern or a dataset. We usually reserve the term *dataset* for a k-dimensional vector set that represents some complete set of data that we are interested in processing. We usually reserve the term *pattern* for a k-dimensional vector set that is a subset of some specified dataset or a transformation of some subset of a dataset. Also, if we have two k-dimensional vector sets P and D and we wish to search for occurrences of P in D then we would usually refer to P as a pattern and D as a dataset.

There are many different appropriate ways of representing a passage of music as a multidimensional dataset. Figure 8 shows a 5-dimensional dataset that represents the music in Figure 7. Each datapoint in Figure 8 represents a single note in Figure 7. The first element in each datapoint represents the onset time of the note in terms of the number of semiquavers that have elapsed by the time the note occurs. The second element in each datapoint represents the *chromatic pitch* of the note as defined by Meredith (1999, 2001). The chromatic pitch of a note can be understood to be a numerical representation of the key on a normal piano keyboard that would have to be pressed to play the note. The chromatic pitch of $A\natural_0$ which is usually the lowest note on a piano keyboard, is defined to be 0. The chromatic pitch of $B\flat_0$, the note one semitone above $A\natural_0$, is therefore 1 and the chromatic pitch of middle C ($C\natural_4$) is 39. An increase in pitch of one semitone results in an increase of 1 in chromatic pitch. The concept of chromatic pitch is similar to Brinkman's *continuous pitch code* (Brinkman, 1990, p. 122). The third element in each datapoint represents the *morphetic pitch* of the note as defined by Meredith (1999, 2001). The morphetic pitch of a note indicates the position of the notehead of the note on the staff—a rise of one step on a staff increases the morphetic pitch by one. The morphetic pitch of $A\natural_0$ is defined to be 0. The morphetic pitch of middle C ($C\natural_4$) is therefore 23, D above middle C has a morphetic pitch of 24 and so on. The concept of morphetic pitch is similar to Brinkman's *continuous name code* (Brinkman, 1990, p. 126). The fourth element in each datapoint represents the duration of the note measured in semiquavers and the fifth element represents the voice in which the note occurs (here, a value of 1 indicates the upper voice and a value of 2 indicates the lower voice).

PLEASE INSERT FIGURE 7 ABOUT HERE.

PLEASE INSERT FIGURE 8 ABOUT HERE.

Each orthogonal projection of a multidimensional dataset is itself a multidimensional dataset. This implies that if one has a rich multidimensional dataset representation D of a musical passage and a repetition-discovery algorithm such as `SIA` or `SIATEC` that can process datasets of any dimensionality, then one is equipped for discovering repeated structures not only in the original complete dataset D but also any orthogonal projection of D. This can be useful. For example, it is undoubtedly true that most listeners would consider patterns B and C in Figure 7 to be perceptually significant repetitions of pattern A. If one considers only the first two elements in each datapoint in Figure 8 then one gets the projection shown in Figure 9 which represents the onset time and chromatic pitch of each note. We say that two patterns P_1 and P_2 are *translationally equivalent* if and only if there exists a vector $\mathbf{v}$ such that P_1 translated by $\mathbf{v}$ gives P_2. Note that patterns B and C in Figure 9 (which correspond to patterns B and C respectively in Figure 7) are *not* translationally equivalent to A. This implies that an algorithm such as `SIA` or `SIATEC` that only discovers instances of repetition in which the two patterns are translationally equivalent, would not discover the repetitions indicated in Figure 9. However, if one considers only the first and third element in each datapoint in Figure 8 then one gets the 2-dimensional projection shown in Figure 10. Patterns A, B and C in Figure 10 *are* translationally equivalent and therefore would be discovered by an algorithm that only finds instances of repetition in which the repeated patterns are translationally equivalent. This example also shows that instances of approximate repetition in one projection of a dataset might correspond to instances of exact repetition in some other projection of the dataset.

$\boxed{\text{PLEASE INSERT FIGURE 9 ABOUT HERE.}}$

$\boxed{\text{PLEASE INSERT FIGURE 10 ABOUT HERE.}}$

Multidimensional datasets can also be used to represent multi-channel digital audio data. For example, a PCM audio file can be represented as a dataset in which each datapoint represents a single sample as an ordered triple in which the first element represents time, the second represents sample magnitude and the third represents the audio channel. Indeed, the only difference between such a representation and the way that this information is typically represented in uncompressed PCM audio files such as AIFF and WAV format files, is that in the latter, to save space, the time and channel of each sample are represented implicitly by the position of the sample within the file.

A time-varying audio frequency spectrum can be represented as a multidimensional dataset in which each datapoint is an ordered triple such that the first element represents time, the second represents frequency or log-frequency and the third represents amplitude or power. If log-frequency is chosen for the second dimension, tones with similar timbres (and thus similar spectra) correspond to approximately translationally equivalent patterns in the two-dimensional projection of the representation onto the plane containing the log-frequency and time axes. This suggests that an algorithm such as `SIATEC` that discovers sets of translationally equivalent patterns might find use in a system for separating out instrumental parts in digital audio data (cf. Tanguiane, 1993, p. 16–18).

The algorithms described below can be used to process any multidimensional dataset whatsoever regardless of whether the dataset represents a digital audio recording, a score or 'performance activity information' (Huron, 1992, p. 13) such as MIDI data derived from a human performance on a MIDI-enabled instrument. In this paper, however, we focus on the problem of discovering repeated patterns in scores and 'piano-roll'-like representations.

5 The functions computed by `SIA` and `SIATEC`

In this section we develop mathematical expressions for the functions computed by the `SIA` and `SIATEC` algorithms. Let D be a dataset and let $\mathbf{d_1}$ and $\mathbf{d_2}$ be any two datapoints in D. The vector from $\mathbf{d_1}$ to $\mathbf{d_2}$ is given by $\mathbf{d_2}-\mathbf{d_1}$ where the minus sign denotes vector subtraction. For example, in the dataset in Figure 11, the vector from $\mathbf{a} = \langle 1,1\rangle$ to $\mathbf{f} = \langle 3,2\rangle$ is $\langle 3,2\rangle - \langle 1,1\rangle = \langle 3-1, 2-1\rangle = \langle 2,1\rangle$. If $\mathbf{v} = \mathbf{d_2} - \mathbf{d_1}$ then $\mathbf{d_2} = \mathbf{v} + \mathbf{d_1}$ which expresses the fact that the datapoint $\mathbf{d_1}$ can be translated by the vector $\mathbf{v}$ to give the datapoint $\mathbf{d_2}$. If $\mathbf{A}$ is an ordered set or a vector then we denote the cardinality of $\mathbf{A}$ by $|\mathbf{A}|$ and the ith element of $\mathbf{A}$ by $\mathbf{A}[i]$. If $\mathbf{u}$ and $\mathbf{v}$ are two vectors such that $|\mathbf{u}| = |\mathbf{v}| = k$ then we say that $\mathbf{u}$ is *less than* $\mathbf{v}$, denoted by $\mathbf{u} < \mathbf{v}$, if and only if there exists an integer i such that $1 \leq i \leq k$ and $\mathbf{u}[i] < \mathbf{v}[i]$ and $\mathbf{u}[j] = \mathbf{v}[j]$ for $1 \leq j < i$. For example, $\langle 1,1\rangle < \langle 1,2\rangle < \langle 2,1\rangle$.

PLEASE INSERT FIGURE 11 ABOUT HERE.

We say that a pattern P is *translatable* by a vector $\mathbf{v}$ in a dataset D if and only if P can be translated by $\mathbf{v}$ to give a pattern that is a subset of D. For example, in the dataset shown in Figure 11, the pattern $\{\mathbf{a},\mathbf{c}\}$ is only translatable by the vectors $\langle 1,1\rangle$ and $\langle 0,2\rangle$.

The *maximal translatable pattern* (MTP) for a vector $\mathbf{v}$ in a dataset D, denoted by $MTP(\mathbf{v},D)$, is the largest pattern translatable by $\mathbf{v}$ in D. Formally,

$$MTP(\mathbf{v},D) = \{\mathbf{d} \,|\, \mathbf{d} \in D \wedge \mathbf{d}+\mathbf{v} \in D\}\,. \tag{1}$$

For example, if we consider the dataset in Figure 11, then the MTP for the vector $\langle 1,0\rangle$ is $\{\mathbf{a},\mathbf{b},\mathbf{d}\}$ and the MTP for $\langle 1,1\rangle$ is $\{\mathbf{a},\mathbf{c}\}$.

In music, MTPs often correspond to the patterns involved in perceptually significant repetitions. For example, each of the patterns A, B, C and D in Figure 12 corresponds to an MTP in the two-dimensional dataset representation shown in Figure 13. Our goal in developing `SIA` was to design an efficient algorithm for computing all the non-empty MTPs in a dataset. The MTP for a vector $\mathbf{v}$ in a dataset D is non-empty if and only if there exist at least two datapoints $\mathbf{d_1}$ and $\mathbf{d_2}$ in D such that $\mathbf{v} = \mathbf{d_2} - \mathbf{d_1}$. This implies that the complete set of non-empty MTPs for a dataset D is given by

$$\mathcal{P}(D) = \{MTP(\mathbf{d_2}-\mathbf{d_1},D) \,|\, \mathbf{d_1},\mathbf{d_2} \in D\}\,. \tag{2}$$

PLEASE INSERT FIGURE 12 ABOUT HERE.

PLEASE INSERT FIGURE 13 ABOUT HERE.

In the dataset in Figure 13, pattern A is the MTP for the vector $\langle 16,-12\rangle$. If we translate pattern A by the vector $\langle 16,-12\rangle$ we get pattern D which is the MTP for the vector $\langle -16,12\rangle$. Let us denote by $\tau(P,\mathbf{v})$ the pattern that results when the pattern P is translated by the vector $\mathbf{v}$. Formally,

$$\tau(P,\mathbf{v}) = \{\mathbf{d}+\mathbf{v} \,|\, \mathbf{d} \in P\}\,. \tag{3}$$

We will now prove that, in general, if the MTP for $\mathbf{v}$ is translated by $\mathbf{v}$, the resulting pattern is the MTP for the vector $-\mathbf{v}$.

Lemma 1 *If D is a dataset and $\mathbf{v}$ is a vector then*

$$\tau(MTP(\mathbf{v},D),\mathbf{v}) = MTP(-\mathbf{v},D). \tag{4}$$

Proof

From Eq.3 we deduce that

$$\tau(MTP(\mathbf{v},D),\mathbf{v}) = \{\mathbf{d_1}+\mathbf{v} \,|\, \mathbf{d_1} \in MTP(\mathbf{v},D)\}\,. \tag{5}$$

Substituting Eq.1 into Eq.5, we find that

$$\begin{aligned}\tau(MTP(\mathbf{v},D),\mathbf{v}) &= \{\mathbf{d_1}+\mathbf{v} \,|\, \mathbf{d_1} \in \{\mathbf{d_2} \,|\, \mathbf{d_2} \in D \wedge \mathbf{d_2}+\mathbf{v} \in D\}\} \\ &= \{\mathbf{d_2}+\mathbf{v} \,|\, \mathbf{d_2} \in D \wedge \mathbf{d_2}+\mathbf{v} \in D\}\,. \end{aligned} \tag{6}$$

If we let $\mathbf{d_3} = \mathbf{d_2} + \mathbf{v}$ and substitute this into Eq.6, we deduce that

$$\tau(\mathit{MTP}(\mathbf{v}, D), \mathbf{v}) = \{\mathbf{d_3} \,|\, \mathbf{d_3} - \mathbf{v} \in D \wedge \mathbf{d_3} \in D\}\,. \tag{7}$$

Eqs.7 and 1 together imply

$$\tau(\mathit{MTP}(\mathbf{v}, D), \mathbf{v}) = \mathit{MTP}(-\mathbf{v}, D).$$

■

Lemma 1 tells us that if we compute $\mathit{MTP}(\mathbf{d_2} - \mathbf{d_1}, D)$ then we can find $\mathit{MTP}(\mathbf{d_1} - \mathbf{d_2}, D)$ simply by translating $\mathit{MTP}(\mathbf{d_2} - \mathbf{d_1}, D)$ by $\mathbf{d_2} - \mathbf{d_1}$. It is also clear that $\mathit{MTP}(\mathbf{0}, D) = D$ where $\mathbf{0}$ is the zero vector. These two facts imply that our MTP-discovery algorithm only really needs to compute the set

$$\mathcal{P}'(D) = \{\mathit{MTP}(\mathbf{d_2} - \mathbf{d_1}, D) \,|\, \mathbf{d_1}, \mathbf{d_2} \in D \wedge \mathbf{d_1} < \mathbf{d_2}\}\,. \tag{8}$$

However, if `SIA` simply generated the set $\mathcal{P}'(D)$, then it would not be possible to determine the vector for which any given pattern in $\mathcal{P}'(D)$ was the MTP. Therefore, `SIA` actually computes the set

$$\mathcal{S}(D) = \{\langle \mathbf{d_2} - \mathbf{d_1}, \mathit{MTP}(\mathbf{d_2} - \mathbf{d_1}, D)\rangle \,|\, \mathbf{d_1}, \mathbf{d_2} \in D \wedge \mathbf{d_1} < \mathbf{d_2}\}. \tag{9}$$

Each member of $\mathcal{S}(D)$ is an ordered pair in which the first element is a vector $\mathbf{v}$ and the second element is the MTP for $\mathbf{v}$ in D. Figure 14 shows $\mathcal{S}(D)$ for the dataset in Figure 11.

PLEASE INSERT FIGURE 14 ABOUT HERE.

Figure 16 is a 2-dimensional dataset representation of the music in Figure 15. In this representation only the morphetic pitch and onset time of each note event are represented. Patterns A–H in Figure 15 correspond to patterns A–H in Figure 16. `SIA` tells us that pattern A in Figure 16 is the MTP in this dataset for the vector $\langle 28, -10\rangle$. In other words, `SIA` discovers that a repetition of pattern A in Figure 15 occurs 28 semiquavers later, diatonically transposed down an eleventh (pattern H). However, `SIA` tells us nothing about all the other transposed occurrences of pattern A (i.e., patterns B–G) in this two-bar passage. Ideally, we would like to know not only that pattern A in Figure 16 is an MTP but also all the other patterns in the dataset that are translationally equivalent to A.

PLEASE INSERT FIGURE 15 ABOUT HERE.

PLEASE INSERT FIGURE 16 ABOUT HERE.

If P is a pattern in a dataset D then we say that the *translational equivalence class* (TEC) of P in D, denoted by $\mathit{TEC}(P, D)$, is the set that contains all and only those patterns in D that are translationally equivalent to P. For example, in Figure 16 the TEC of pattern A is the set of patterns labelled A–H. Formally, we say that two patterns P_1 and P_2 are translationally equivalent, denoted by $P_1 \equiv_\tau P_2$, if and only if there exists a vector $\mathbf{v}$ such that $\tau(P_1, \mathbf{v}) = P_2$. The TEC of a pattern P in a dataset D is then given by

$$\mathit{TEC}(P, D) = \{Q \,|\, Q \equiv_\tau P \wedge Q \subseteq D\}\,. \tag{10}$$

Our goal in developing `SIATEC` was to design an efficient algorithm for computing the set

$$\mathcal{T}(D) = \{ \mathit{TEC}(\mathit{MTP}(\mathbf{d_2} - \mathbf{d_1}, D), D) \,|\, \mathbf{d_1}, \mathbf{d_2} \in D \} . \tag{11}$$

It can be shown that the translational equivalence relation is a true equivalence relation in the mathematical sense—that is, it is reflexive, transitive and symmetric (Meredith *et al.*, 2001). This implies that the translational equivalence relation partitions the power set of a dataset into translational equivalence classes. This means that every pattern in a dataset is a member of exactly one TEC. However, from Lemma 1 we know that

$$\tau(\mathit{MTP}(\mathbf{d_2} - \mathbf{d_1}, D), \mathbf{d_2} - \mathbf{d_1}) = \mathit{MTP}(\mathbf{d_1} - \mathbf{d_2}, D).$$

Therefore

$$\mathit{TEC}(\mathit{MTP}(\mathbf{d_2} - \mathbf{d_1}, D), D) = \mathit{TEC}(\mathit{MTP}(\mathbf{d_1} - \mathbf{d_2}, D), D).$$

Moreover, we know that $\mathit{MTP}(\mathbf{0}, D) = D$ and therefore $\mathit{TEC}(\mathit{MTP}(\mathbf{0}, D), D) = \{D\}$ which is a trivial translational equivalence class. Therefore, instead of computing $\mathcal{T}(D)$ as defined in Eq.11, `SIATEC` actually computes the set

$$\mathcal{T}'(D) = \{ \mathit{TEC}(\mathit{MTP}(\mathbf{d_2} - \mathbf{d_1}, D), D) \,|\, \mathbf{d_1}, \mathbf{d_2} \in D \wedge \mathbf{d_1} < \mathbf{d_2} \}. \tag{12}$$

It can be shown that $\mathcal{T}(D) = \mathcal{T}'(D) \cup \{\{D\}\}$ (Meredith *et al.*, 2001).

If P is a pattern in a dataset D then we say that $\mathbf{v}$ is a *translator* of P in D if and only if P is translatable by $\mathbf{v}$ in D. The *set of translators* for P in D, which we denote by $T(P, D)$, is the set that only contains all vectors by which P is translatable. Formally,

$$T(P, D) = \{\mathbf{v} \,|\, \tau(P, \mathbf{v}) \subseteq D\} . \tag{13}$$

For example, in Figure 11 the set of translators for the pattern $\{\mathbf{a}, \mathbf{c}\}$ is $\{\langle 0, 0\rangle , \langle 1, 1\rangle , \langle 0, 2\rangle\}$ and the set of translators for the pattern $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$ is $\{\langle 0, 0\rangle , \langle 1, 0\rangle\}$.

The TEC of a pattern P in a dataset D can therefore be represented efficiently by the ordered pair $\langle P, T(P, D)\rangle$. For any given TEC, E, there are $|E|$ such representations, one for each pattern in E. In general, this ordered-pair representation for a TEC can be much more space-efficient than explicitly writing out every member pattern of the TEC in full. For example, if there are 20 patterns in a dataset that are translationally equivalent to a pattern P_1 containing 10 datapoints then printing out the TEC for P_1 in full would involve printing 200 datapoints. However, if this TEC were represented as the ordered pair $\langle P, T(P, D)\rangle$ then only $10+20 = 30$ vectors would need to be printed. Incidentally, this example illustrates how an algorithm such as `SIATEC` that discovers TECs can be used as the basis of an algorithm for compressing multidimensional data.

In the output of `SIATEC`, each distinct TEC, E, in $\mathcal{T}'(D)$ is therefore represented as an ordered pair $\langle P, T(P, D)\rangle$ where P is a member of E and $T(P, D)$ is the set of translators for P in D. Figure 17 shows $\mathcal{T}'(D)$ for the dataset shown in Figure 11.

PLEASE INSERT FIGURE 17 ABOUT HERE.

6 The `SIA` and `SIATEC` algorithms

In this section we describe the `SIA` and `SIATEC` algorithms. Meredith *et al.* (2001) provide a more complete description and analysis of these algorithms.

6.1 The `SIA` algorithm

When given a multidimensional dataset, D, as input, `SIA` efficiently computes $\mathcal{S}(D)$ as defined in Eq.9 above. For a k-dimensional dataset containing n datapoints, the worst-case running time of `SIA` is $O(kn^2 \log_2 n)$ and its worst-case space complexity is $O(kn^2)$. The algorithm consists of the following four steps.

`SIA`: Step 1 – Sorting the dataset The first step in `SIA` is to sort the dataset D to give an ordered set $\mathbf{D}$ that contains all and only the datapoints in the dataset in increasing order. For the dataset in Figure 11, the result of this first step would be the ordered set

$$\mathbf{D} = \langle \langle 1,1\rangle , \langle 1,3\rangle , \langle 2,1\rangle , \langle 2,2\rangle , \langle 2,3\rangle , \langle 3,2\rangle\rangle . \tag{14}$$

For a k-dimensional dataset of size n, this can be done using merge sort (Cormen *et al.*, 1990, pp. 12–15) in a worst-case running time of $O(kn \log_2 n)$.[3]

`SIA`: Step 2 – Computing the vector table The second step in `SIA` is to compute the set

$$V = \{\langle \mathbf{D}[j] - \mathbf{D}[i], i\rangle \mid 1 \leq i < j \leq |\mathbf{D}|\} . \tag{15}$$

Note that each member of V is an ordered pair in which the first element is the vector from datapoint $\mathbf{D}[i]$ to datapoint $\mathbf{D}[j]$ and the second element is the index of the 'origin' datapoint, $\mathbf{D}[i]$, in $\mathbf{D}$. For the dataset in Figure 11, V contains all the elements below the leading diagonal in Table 1.

PLEASE INSERT TABLE 1 ABOUT HERE.

We call a table like the one in Table 1 a *vector table*. Each element in this table is an ordered pair $\langle \mathbf{v}, i\rangle$ where i gives the number of the column in which the element occurs and $\mathbf{v}$ is the vector *from* the datapoint at the head of the column in which the element occurs *to* the datapoint at the head of the row in which the element occurs. For a k-dimensional dataset of size n, this second step of the algorithm involves computing $\frac{n(n-1)}{2}$ vector subtractions. It can be accomplished in a worst-case running time of $O(kn^2)$.

[3] When merge sort is implemented using arrays, it requires linear extra memory and the additional work spent copying to and from the temporary array throughout the algorithm has the effect of slowing down the sort considerably. However, we use a special implementation of merge sort that employs linked lists and in this implementation no extra memory is required and no copying of data is performed.

SIA: Step 3 – Sorting the vectors If $\langle \mathbf{u}, i \rangle$ and $\langle \mathbf{v}, j \rangle$ are any two elements in the set V computed in the second step of the algorithm (Eq.15) then we define that $\langle \mathbf{u}, i \rangle$ is *less than* $\langle \mathbf{v}, j \rangle$, denoted by $\langle \mathbf{u}, i \rangle < \langle \mathbf{v}, j \rangle$, if and only if $\mathbf{u} < \mathbf{v}$ or $\mathbf{u} = \mathbf{v}$ and $i < j$.

The third step in SIA is to sort V to give an ordered set $\mathbf{V}$ that contains the elements of V in increasing order. For example, the column headed $\mathbf{V}[i]$ in Table 2 gives $\mathbf{V}$ for the dataset in Figure 11. An examination of Table 1 reveals that the vectors increase as one descends a column and decrease as one goes from left to right along a row. In our implementation of SIA we use a two-dimensional linked list to represent V as a vector table like the one in Table 1. We then use a modified version of merge sort, that exploits the fact that the columns and rows in this vector table are already sorted, to accomplish this third step of the algorithm more rapidly than would be achievable using plain merge sort on the completely unsorted set V. The worst-case running time of this step of the algorithm is $O(kn^2 \log_2 n)$.

PLEASE INSERT TABLE 2 ABOUT HERE.

SIA: Step 4 – Printing out $\mathcal{S}(D)$ If $\mathbf{A}$ is an ordered set of ordered sets then $\mathbf{A}[i, j]$ denotes the jth element of the ith element of $\mathbf{A}$. For example, if $\mathbf{A} = \langle \langle a, b, c \rangle, \langle d, e \rangle, \langle f \rangle \rangle$ then $\mathbf{A}[1, 3] = c$, $\mathbf{A}[2, 1] = d$ and $\mathbf{A}[3, 1] = f$. As pointed out above, the column headed $\mathbf{V}[i]$ in Table 2 gives $\mathbf{V}$ for the dataset in Figure 11. For each of these ordered pairs, $\mathbf{V}[i]$, the datapoint $\mathbf{D}[\mathbf{V}[i, 2]]$ is printed next to it in the third column in Table 2. For example, $\mathbf{V}[1] = \langle \langle 0, 1 \rangle, 3 \rangle$ in Table 2, so $\mathbf{V}[1, 2] = 3$ and $\mathbf{D}[\mathbf{V}[1, 2]] = \langle 2, 1 \rangle$, the third datapoint in the ordered set $\mathbf{D}$ for the dataset shown in Figure 11.

As indicated on the right-hand side of the third column in Table 2, the MTP for a vector $\mathbf{v}$ is the set of consecutive datapoints $\mathbf{D}[\mathbf{V}[i, 2]]$ in the third column that corresponds to the set of consecutive ordered pairs $\mathbf{V}[i]$ in the second column for which $\mathbf{V}[i, 1] = \mathbf{v}$. The complete set $\mathcal{S}(D)$ as defined in Eq.9 can be printed out using the algorithm in Figure 18. In our pseudocode, block structure is indicated by indentation and the symbol '$\leftarrow$' indicates assignment. Figure 19 shows the output generated by this algorithm for the dataset in Figure 11.

PLEASE INSERT FIGURE 18 ABOUT HERE.

PLEASE INSERT FIGURE 19 ABOUT HERE.

SIA discovers the set $\mathcal{P}'(D)$ of non-empty MTPs defined in Eq.8 and from Table 2 it can easily be seen that SIA accomplishes this simply by sorting the set V defined in Eq.15. It is clear from Table 1 that, for a dataset of size n, the number of elements in V is $\frac{n(n-1)}{2}$. Therefore, if we use P to denote an MTP in $\mathcal{P}'(D)$,

$$\sum_{P \in \mathcal{P}'(D)} |P| = \frac{n(n-1)}{2}.$$

Therefore the total number of vectors that have to be printed when $\mathcal{S}(D)$ is printed is $\frac{n(n-1)}{2}$ plus one vector for each MTP in $\mathcal{P}'(D)$. Since $|\mathcal{P}'(D)| \leq \frac{n(n-1)}{2}$, the total number of vectors to be printed out is certainly less than or equal to $n(n-1)$. Therefore, for a k-dimensional dataset containing n datapoints, $\mathcal{S}(D)$ can be printed out in a worst-case running time of $O(kn^2)$.

6.2 The `SIATEC` algorithm

When given a multidimensional dataset, D, as input, our second algorithm, `SIATEC`, efficiently computes $\mathcal{T}'(D)$ as defined in Eq.12 above. For a k-dimensional dataset containing n datapoints, the worst-case running time of `SIATEC` is $O(kn^3)$ and its worst-case space complexity is $O(kn^2)$. The algorithm consists of the following seven steps.

`SIATEC`: Step 1 – Sorting the dataset This is exactly the same as Step 1 of `SIA` as described on page 17 above.

`SIATEC`: Step 2 – Computing W The second step in `SIATEC` is to compute the ordered set of ordered sets

$$\mathbf{W} = \langle\langle \mathbf{W}[1,1], \ldots \mathbf{W}[1,|\mathbf{D}|]\rangle, \ldots \langle \mathbf{W}[|\mathbf{D}|,1], \ldots \mathbf{W}[|\mathbf{D}|,|\mathbf{D}|]\rangle\rangle$$

where

$$\mathbf{W}[i,j] = \langle \mathbf{D}[j] - \mathbf{D}[i], i\rangle \,. \tag{16}$$

$\mathbf{W}$ can be visualized as a vector table like Table 3 (which shows $\mathbf{W}$ for the dataset in Figure 11). The only difference between the vector table ($\mathbf{W}$) computed by `SIATEC` and the vector table computed in Step 2 of `SIA` (see Table 1) is that in `SIATEC`, all the elements in this table are filled in; whereas in `SIA`, only those elements below the leading diagonal are computed. Computing the whole table rather than just the region below the leading diagonal allows us to compute more efficiently the set of translators for each MTP (see Step 7 below).

PLEASE INSERT TABLE 3 ABOUT HERE.

Computing $\mathbf{W}$ for a k-dimensional dataset of size n involves computing n^2 vector subtractions. Each of these vector subtractions involves carrying out k scalar subtractions so the overall worst-case running time of this step is $O(kn^2)$.

`SIATEC`: Step 3 – Computing V The third step of `SIATEC` is to compute the set V as defined in Eq.15. This is the same set as that computed in Step 2 of `SIA`. In `SIATEC`, V can be computed directly from $\mathbf{W}$ since

$$V = \{\mathbf{W}[i,j] \mid 1 \leq i < j \leq |\mathbf{D}|\} \,. \tag{17}$$

In fact, in our implementation of `SIATEC`, V is computed at the same time as $\mathbf{W}$.

`SIATEC`: Step 4 – Sorting V to produce V This step is exactly the same as Step 3 of `SIA`.

`SIATEC`: Step 5 – 'Vectorizing' the MTPs $\mathbf{V}$ is effectively a sorted representation of $\mathcal{S}(D)$ (Eq.9) (see Step 4 of `SIA` and Table 2). The purpose of `SIATEC` is to compute $\mathcal{T}'(D)$ (Eq.12) which is the set that only contains every TEC that is the TEC of an MTP in $\mathcal{P}'(D)$. $\mathcal{P}'(D)$ can be obtained from $\mathbf{V}$ but it is possible for two or more MTPs in $\mathcal{P}'(D)$ to be translationally equivalent. For example, the MTPs in the dataset in Figure 11 for the vectors $\langle 0,2\rangle$, $\langle 1,-1\rangle$ and $\langle 1,1\rangle$ are translationally equivalent (see Table 2). If two patterns are translationally equivalent then they are members of the same TEC. Therefore, if we naïvely compute the TEC of each MTP in $\mathcal{P}'(D)$, we run the risk of computing the same TEC more than once which is inefficient. We therefore partition $\mathcal{P}'(D)$ into translational equivalence classes and then select just one MTP from each of these classes, discarding the others.

If P is a pattern then let $\mathit{SORT}(P)$ be the function that returns the ordered set that only contains all the datapoints in P sorted into increasing order. If $\mathbf{P}$ is an ordered set of datapoints then let $\mathit{VEC}(\mathbf{P})$ be the function that returns the ordered set of vectors

$$\mathit{VEC}(\mathbf{P}) = \langle \mathbf{P}[2] - \mathbf{P}[1], \mathbf{P}[3] - \mathbf{P}[2], \ldots \mathbf{P}[|P|] - \mathbf{P}[|P| - 1] \rangle. \tag{18}$$

If P_1 and P_2 are two patterns in a dataset, then

$$\mathit{VEC}(\mathit{SORT}(P_1)) = \mathit{VEC}(\mathit{SORT}(P_2)) \iff P_1 \equiv_\tau P_2. \tag{19}$$

We say that $\mathit{VEC}(\mathit{SORT}(P))$ is the *vectorized representation* of the pattern P. In the ordered set $\mathbf{V}$ computed in step 4 of `SIATEC`, each MTP, P, is represented in its sorted form as $\mathit{SORT}(P) = \mathbf{P}$ (see Table 2). Therefore, if we want to use Eq.19 to partition $\mathcal{P}'(D)$ we first have to compute $\mathit{VEC}(\mathbf{P})$ for each of the sorted MTPs, $\mathbf{P}$, in $\mathbf{V}$. Step 5 of `SIATEC` is therefore to compute

$$X = \{\langle i, \mathit{VEC}(\mathit{SORT}(P))\rangle \mid \langle \mathbf{v}, P\rangle \in \mathcal{S}(D) \wedge \mathbf{V}[i,1] = \mathbf{v} \wedge (i = 1 \vee \mathbf{V}[i-1,1] \neq \mathbf{v})\}. \tag{20}$$

If $\mathbf{V}[i]$ and $\mathbf{V}[j]$ are two distinct elements of $\mathbf{V}$ and $\mathbf{V}[i] < \mathbf{V}[j]$ but $\mathbf{V}[i,1] = \mathbf{V}[j,1]$ (i.e., the vectors in $\mathbf{V}[i]$ and $\mathbf{V}[j]$ are the same) then $\mathbf{V}[i,2] < \mathbf{V}[j,2]$ which implies that $\mathbf{D}[\mathbf{V}[i,2]] < \mathbf{D}[\mathbf{V}[j,2]]$. This means that the datapoints within each MTP in the $\mathbf{V}$ representation of $\mathcal{S}(D)$ are sorted in increasing order, as can be seen in the output of `SIA` (Figure 19) generated by the algorithm in Figure 18.

X can be efficiently computed directly from $\mathbf{V}$ and $\mathbf{D}$ using the algorithm in Figure 20 which exploits the fact that the MTPs in $\mathbf{V}$ are already sorted. In Figure 20, the set X is actually represented as an ordered set $\mathbf{X}$. When the algorithm in Figure 20 has terminated, the ordered set $\mathbf{X}$ only contains all the elements of X (with no duplicates). In Figure 20, $\langle\,\rangle$ denotes the empty ordered set and the symbol $\oplus$ denotes concatenation: if A and B are two ordered sets such that $A = \langle a_1, \ldots a_m\rangle$ and $B = \langle b_1, \ldots b_n\rangle$ then

$$A \oplus B = \langle a_1, \ldots a_m, b_1, \ldots b_n\rangle\,.$$

PLEASE INSERT FIGURE 20 ABOUT HERE.

Figure 21 shows the state of $\mathbf{X}$ for the dataset in Figure 11 at the termination of Step 5 of `SIATEC`. For a k-dimensional dataset of size n, the worst-case running time of the algorithm in Figure 20 is $O(kn^2)$.

PLEASE INSERT FIGURE 21 ABOUT HERE.

`SIATEC`: **Step 6 – Sorting X** Let $\mathbf{Q}_1$ and $\mathbf{Q}_2$ be any two ordered sets in which each element is a k-dimensional vector. We define that $\mathbf{Q}_1$ is *less than* $\mathbf{Q}_2$, denoted by $\mathbf{Q}_1 < \mathbf{Q}_2$ if and only if one of the following two conditions is satisfied:

1. $|\mathbf{Q}_1| < |\mathbf{Q}_2|$.
2. $|\mathbf{Q}_1| = |\mathbf{Q}_2|$ and there exists an integer $1 \leq i \leq |\mathbf{Q}_1|$ such that $\mathbf{Q}_1[i] < \mathbf{Q}_2[i]$ and $\mathbf{Q}_1[j] = \mathbf{Q}_2[j]$ for all $1 \leq j < i$.

(See page 13 for a definition of the expression $\mathbf{u} < \mathbf{v}$ when $\mathbf{u}$ and $\mathbf{v}$ are vectors.) In Step 6 of `SIATEC`, the ordered set $\mathbf{X}$ generated by the algorithm in Figure 20 is sorted to produce the ordered set $\mathbf{Y}$ which satisfies the following two conditions:

1. $\mathbf{Y}$ only contains all the elements of $\mathbf{X}$.
2. If $\mathbf{Y}[i]$ and $\mathbf{Y}[j]$ are any two distinct elements of $\mathbf{Y}$ then $i < j$ if and only if

$$\mathbf{Y}[i,2] < \mathbf{Y}[j,2] \vee (\mathbf{Y}[i,2] = \mathbf{Y}[j,2] \wedge \mathbf{Y}[i,1] < \mathbf{Y}[j,1]).$$

Figure 22 shows $\mathbf{Y}$ for the dataset in Figure 11. For a k-dimensional dataset of size n, this step of the algorithm can be accomplished in a worst-case running time of $O(kn^2 \log_2 n)$ using merge sort.

PLEASE INSERT FIGURE 22 ABOUT HERE.

We know that

$$\mathit{MTP}(\mathbf{V}[\mathbf{Y}[i,1],1],D) \equiv_\tau \mathit{MTP}(\mathbf{V}[\mathbf{Y}[j,1],1],D) \iff \mathbf{Y}[i,2] = \mathbf{Y}[j,2].$$

So Figure 22 tells us, for example, that the MTPs for the vectors $\mathbf{V}[3,1] = \langle 0,2\rangle$, $\mathbf{V}[6,1] = \langle 1,-1\rangle$ and $\mathbf{V}[11,1] = \langle 1,1\rangle$ are translationally equivalent since the vectorized representation of each of these patterns is $\langle\langle 1,0\rangle\rangle$. This implies that we only have to compute the TEC of one of these patterns and the other two can be disregarded.

`SIATEC`: **Step 7 – Printing out $\mathcal{T}'(D)$** The final step of `SIATEC` is to print out $\mathcal{T}'(D)$. This can be done using the algorithm in Figure 23. Recall that each TEC in $\mathcal{T}'(D)$ is represented as an ordered pair $\langle P, T(P,D)\rangle$ where P is an MTP and $T(P,D)$ is the set of translators for P in the dataset D (see Eq.13 and discussion on page 16 above). In Figure 23, each MTP is printed out using the algorithm `PRINT_PATTERN` called in line 14. This algorithm is given in Figure 24.

PLEASE INSERT FIGURE 23 ABOUT HERE.

PLEASE INSERT FIGURE 24 ABOUT HERE.

The set of translators for each TEC is printed out using the algorithm `PRINT_SET_OF_TRANSLATORS` called in line 16 of Figure 23. This algorithm, which is given in Figure 25, exploits the fact that

$$T(\{\mathbf{D}[i]\}, D) = \bigcup_{j=1}^{|D|} \{\mathbf{W}[i,j,1]\}.$$

That is, the set of translators for a datapoint $\mathbf{D}[i]$ is the set that only contains every vector that occurs as the first element in an ordered pair in the ith column in the vector table computed in Step 2 of `SIATEC` (see Table 3). In Figure 23, each MTP is represented as a set of indices, $\mathbf{I}$: the pattern represented by $\mathbf{I}$ is simply $\{\mathbf{D}[i] \mid i \in \mathbf{I}\}$. The set of translators for the pattern represented by $\mathbf{I}$ is therefore

$$\bigcap_{i\in\mathbf{I}} T(\{\mathbf{D}[i]\}, D) = \bigcap_{i\in\mathbf{I}} \left(\bigcup_{j=1}^{|D|} \{\mathbf{W}[i,j,1]\} \right). \tag{21}$$

In other words, the set of translators for a pattern is the set that only contains those vectors that occur in *all* the columns in the vector table corresponding to the datapoints in the pattern. For example, if D is the dataset in Figure 11, the set of translators for the pattern $\{\mathbf{a},\mathbf{c}\} = \{\langle 1,1\rangle, \langle 2,1\rangle\}$ is the set that only contains all the vectors that occur in *both* the first and third columns in Table 3:

$$\begin{aligned} T(\{\langle 1,1\rangle, \langle 2,1\rangle\}, D) &= \{\langle 0,0\rangle, \langle 0,2\rangle, \langle 1,0\rangle, \langle 1,1\rangle, \langle 1,2\rangle, \langle 2,1\rangle\} \\ &\qquad \cap \{\langle -1,0\rangle, \langle -1,2\rangle, \langle 0,0\rangle, \langle 0,1\rangle, \langle 0,2\rangle, \langle 1,1\rangle\} \\ &= \{\langle 0,0\rangle, \langle 0,2\rangle, \langle 1,1\rangle\} \end{aligned}$$

The algorithm `PRINT_SET_OF_TRANSLATORS` is an efficient algorithm for computing the expression on the right-hand side of Eq.21.

PLEASE INSERT FIGURE 25 ABOUT HERE.

Using the algorithms in Figures 23, 24 and 25, Step 7 can be accomplished in a worst-case running time of $O(kn^3)$ for a k-dimensional dataset of size n. Figure 26 shows the output generated by the algorithm in Figure 23 for the dataset in Figure 11.

PLEASE INSERT FIGURE 26 ABOUT HERE.

7 Running `SIA` and `SIATEC` on music data

`SIA` and `SIATEC` have been implemented in ANSI C (Kernighan and Ritchie, 1988), compiled using the GNU C compiler and run on a 500MHz Sparc machine with 1GB RAM. The algorithms have been tested on 52 datasets ranging in dimensionality from 2 to 5 and in size from 6 to nearly 3500 datapoints. (Many of these datasets are multidimensional representations of complete works from J. S. Bach's *Das Wohltemperirte Klavier*.)

Figure 27 shows the running times obtained using our implementation of `SIA` on 33 2-dimensional datasets. The smooth curve in this graph represents the function $t = c_{SIA}kn^2\log_2 n$ where k is the dimensionality of the input dataset (in this case 2) and c_{SIA} is the average over all the test datasets of $\frac{t}{kn^2\log_2 n}$ where t is the running time and n is the size of the input dataset. The results support our claim that the worst-case running time of `SIA` is $O(kn^2\log_2 n)$. This particular implementation takes less than 2 minutes of processor time to analyse a 2-dimensional dataset containing nearly 3500 datapoints (i.e., a piece of music containing 3500 notes).

PLEASE INSERT FIGURE 27 ABOUT HERE.

Figure 28 shows the running times obtained using our implementation of `SIATEC` on the same 2-dimensional datasets. The smooth curve in this graph represents the function $t = c_{SIATEC}kn^3$ where k is again the dimensionality of the input dataset and c_{SIATEC} is the average value over all the test datasets of $\frac{t}{kn^3}$ where t is the running time and n is the size of the input dataset. These results support our claim that the worst-case running time of `SIATEC` is $O(kn^3)$. This particular implementation takes about 13 minutes of processor time to analyse a 2-dimensional dataset containing 2000 datapoints (i.e., a piece of music containing 2000 notes).

PLEASE INSERT FIGURE 28 ABOUT HERE.

8 Isolating perceptually significant repetitions

The power set of a dataset D contains $2^{|D|}$ distinct patterns. On the other hand, the number of MTPs generated by `SIA` for a dataset D is less than $\frac{|D|^2}{2}$. Therefore, for all but the very smallest datasets, the set of non-empty MTPs computed by `SIA` contains only a tiny proportion of all the patterns in the dataset. Moreover, our experiments suggest that many of the patterns involved in perceptually significant repetitions in music are MTPs and are therefore discovered by `SIA` and `SIATEC`. For example, as we have already pointed out (see page 14 above) each of the patterns A, B, C and D in Figure 12 corresponds to an MTP in the two-dimensional dataset representation shown in Figure 13. If the extract in Figure 1 is represented as a two-dimensional dataset in which the dimensions represent chromatic pitch and onset time, then the pattern A1 in this figure is an MTP. If the examples in Figures 2 and 6 are represented in the same way, then the patterns labelled A in these examples are both MTPs. Figure 29 shows the first five bars of Bach's two-part Invention in C major (BWV 772). The set of patterns indicated by boxes in the figure corresponds to one of the TECs discovered by `SIATEC` for this dataset. These examples demonstrate that it is possible to use `SIA` and `SIATEC` to discover perceptually significant musical repetitions that are difficult to find using string-based algorithms.

PLEASE INSERT FIGURE 29 ABOUT HERE.

However, `SIATEC` typically discovers tens of thousands of TECs even in relatively short pieces such as those in Bach's *Das Wohltemperirte Klavier* and usually only a very small proportion of these TECs are perceptually significant or analytically interesting. We are therefore currently experimenting with systems that evaluate the output generated by `SIATEC` and isolate those TECs that correspond to particular classes of perceptually significant repetition.

However, it seems that there are various ways in which a repeated musical structure might be perceptually significant or analytically interesting. In other words, it seems that there are various interesting types of structural feature that a repeated musical pattern might be able to tell us something about. For example, the TEC shown in Figure 29 obviously corresponds to the statements of the opening subject and therefore tells us something interesting about the thematic structure of the music. On the other hand, the largest MTP in the Prelude in C major from Book II of Bach's *Das Wohltemperirte Klavier* tells us that there are 115 occasions in the piece when a note is followed precisely one bar later by a note that is a major second lower. At first, this might not seem significant. However, it reflects the interesting fact that there are many occasions in the piece when a bar-long fragment is immediately repeated one tone lower—an example of the baroque contrapuntal technique known as *sequence*.

This suggests that there is probably no single set of rules or heuristics that is capable of isolating all and only those TECs computed by `SIATEC` that correspond to perceptually significant musical repetitions. We therefore need to develop various algorithms, each one designed to isolate repetitions of a particular type.

We are currently in the early stages of developing algorithms for isolating TECs that contain theme-like or motif-like patterns, since it seems likely that these are the patterns that one would wish to include in the index of a music database. Most of the perceptually significant repeated patterns shown in the musical examples in this paper are theme-like or motif-like (see Figures 1, 2, 6, 7, 12, 15 and 29). Each of these patterns can be unambiguously specified by drawing a horizontal rectangle around it in the score. In other words, each of these theme-like or motif-like patterns contains all the notes that occur within the smallest horizontal rectangle that contains the pattern. In geometry, the smallest hypercuboid with edges parallel to the co-ordinate axes that contains a set of points is called the *bounding box* of that set of points (Cormen *et al.*, 1990, p. 889). In two dimensions, the bounding box of a set of points S is simply the smallest rectangle with edges parallel to the x and y axes that contains S (see Figure 30). We say that a pattern P in a dataset is *bounding-box compact* (BBC) if and only if it contains all the datapoints in the dataset that occur within the bounding box of P. We have designed and implemented an algorithm that when given a dataset D and the set $\mathcal{T}'(D)$ as input, outputs the set

$$\{E \,|\, E \in \mathcal{T}'(D) \wedge (\exists P \,|\, P \in E \wedge P \text{ is BBC})\}\,.$$

That is, it searches the set of MTP TECs generated by `SIATEC` and selects all and only those TECs that contain BBC patterns. The set of patterns shown in Figure 29 corresponds to a TEC that contains a BBC MTP. This algorithm can also be used to isolate repetitions such as those in Figures 2 and 6 in which at least one occurrence of the pattern is compact.

PLEASE INSERT FIGURE 30 ABOUT HERE.

Preliminary observations suggest that compactness is one important feature that is common to most theme-like patterns. However, there are certainly theme-like patterns that are not BBC and there are also BBC patterns that are not theme-like. The MTP labelled A in Figure 13 is an example of a theme-like pattern that is not BBC and the pattern labelled Z in Figure 16 is an example of a BBC pattern that is not theme-like.

Although pattern A in Figure 13 is not BBC, it *does* contain all the points in the dataset that occur within the smallest convex polygon that can be drawn around it. In geometry, the *convex* *hull* of a set Q of points in a two-dimensional Euclidean space is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior (Cormen *et al.*, 1990, p. 898). More generally, the convex hull of a subset A of a real vector space is the intersection of all convex sets containing A (Borowski and Borwein, 1989, p. 123, s.v. **convex hull**). Figure 31 shows the convex hull of a set of points in two dimensions. We say that a pattern P in a dataset is *convex hull compact* if and only if it contains all and only those datapoints in the dataset that occur within the convex hull of P. Cases like pattern A in Figure 13 suggest that it might be interesting to develop an algorithm that searches the MTP TECs generated by `SIATEC` and selects all and only those TECs that contain convex-hull compact patterns. Alternatively one could isolate those TECs that contain temporally compact patterns (see page 3 above).

PLEASE INSERT FIGURE 31 ABOUT HERE.

Instead of simply eliminating those TECs in the output of `SIATEC` that do not satisfy certain conditions, one can, instead, use a set of heuristics to calculate a value Q for each TEC that is designed to be an indication of how theme-like the patterns in the TEC are. One can then sort the TECs into decreasing order of Q and output this sorted list. Q might depend upon factors such as frequency of occurrence, pattern size, overlap between pattern occurrences, compactness and so on.[4] We have designed and implemented algorithms along these lines and preliminary experiments have yielded promising results. For example, we designed an algorithm that computes a value of Q for each TEC based on measures of pattern size, frequency of occurrence, compactness and pattern overlap. When this algorithm was run on a two-dimensional dataset representing the passage in Figure 29, the TEC shown in this figure achieved the highest value of Q. Also, when the same algorithm was run on the dataset shown in Figure 13, the TEC containing pattern A was assigned one of the highest values of Q.

The choice of dataset projection can also have a significant effect on the repetitions that are discovered. For example, if one is only interested in repetitions in which each pattern is wholly contained within a single voice, then it may be possible to exclude many MTPs that do not satisfy this condition simply by using an orthogonal projection of the dataset that contains voicing information.

One important difference between our approach and that adopted in most previous work is that we generate a superset of the interesting repetitions and then use heuristics or algorithms to isolate particular classes of repetition. In previous approaches (e.g., Cambouropoulos, 1998; Conklin and Anagnostopoulou, 2001; Rolland, 1999), the investigators have tried hard to avoid generating unwanted patterns at any stage by using more specific algorithms and less flexible representations. However, our approach allows us to develop any number of different 'filtering methods' to isolate repetitions of any type that can be formally defined. This is a much more flexible approach than using more restrictive representations and algorithms that set severe limits right from the outset on the classes of repetition that can be discovered.

[4]cf. Cambouropoulos's (1998, p. 89) 'Selection Function'.

We are only in the early stages of experimenting with heuristics for isolating particular classes of perceptually significant repetition. However, our preliminary results suggest that by using `SIATEC` in conjunction with carefully chosen heuristics it may be possible to isolate various types of perceptually significant repetition directly from a multidimensional representation of a musical score or performance.

9 Summary

Music analysts and psychologists have stressed that identifying important repetitions in music is an essential part of the process by which an expert listener achieves a rich interpretation of a musical work. However, the vast majority of exact repetitions that occur within a musical work are *not* perceptually significant. Moreover, the class of perceptually significant repetitions is a very diverse set that is difficult to characterise formally.

In previous approaches to repetition discovery in music it has been assumed that the music to be analysed is represented using strings. However, there are certain types of interesting repetition in music that cannot be discovered using these string-based approaches. Also, if one wishes to discover many different types of repetition using a string-based approach, then one typically has to run a variety of algorithms on a multitude of different representations of the music.

We propose a *geometric* approach to repetition discovery in which the music to be analysed is represented as a *multidimensional dataset* (a set of points in a multidimensional Euclidean space). Certain types of perceptually significant musical repetition that cannot be discovered using string-processing algorithms can easily be discovered using appropriately designed algorithms that process multidimensional datasets.

By adopting this new geometric approach, we dispense with the need for using a multitude of different representations because the same repetition discovery algorithms can be run on just a small number of orthogonal projections of a single rich multidimensional representation of a musical passage. This new approach allows complex polyphonic music to be analysed as efficiently as monophonic music and allows for the discovery of repeated patterns in the timbre, dynamic and rhythmic structure of a passage as well as its pitch structure.

We do not propose our approach as a *replacement* for string-based approaches. We only claim that it offers a useful *alternative* approach that can be used to discover types of repeated structure that are hard or even practically impossible to compute using string-based approaches (e.g., polyphonic patterns ‘with gaps’). We believe that our geometric approach can also be used to compute many of the types of repetition that can be discovered using string-based approaches. However, there may be certain classes of repetition (e.g., those in which the patterns can be represented as factors) that can be more efficiently computed using string algorithms.

We introduce the concept of a *maximal translatable pattern* (MTP). Given a multidimensional dataset D a pattern P is the MTP for a vector $\mathbf{v}$ if and only if it is the largest pattern in D that can be translated by $\mathbf{v}$ to give another pattern in D.

We also introduce the concept of a *translational equivalence class* (TEC). Two patterns P_1 and P_2 are *translationally equivalent* if and only if P_2 can be obtained by translating P_1. Given a multidimensional dataset D, and a pattern P, then the TEC of P in D contains all and only those patterns in D that are translationally equivalent to P.

We present two new algorithms: `SIA` and `SIATEC`. `SIA` takes a multidimensional dataset D as input and computes all the non-empty MTPs in D. `SIATEC` takes a dataset D as input and computes the set of TECs that contain non-empty MTPs in D. For a k-dimensional dataset of size n, the worst-case running time of `SIA` is $O(kn^2 \log_2 n)$ and the worst-case running time of `SIATEC` is $O(kn^3)$.

`SIA` and `SIATEC` have been implemented in C and compiled on a variety of platforms. The programs have been run on datasets ranging in dimensionality from 2 to 5 and in size from 6 to nearly 3500 datapoints. When run on a 500MHz Sparc machine, `SIA` takes less than 2 minutes of processor time to analyse a dataset containing 3500 two-dimensional datapoints and `SIATEC` takes around 13 minutes of processor time to analyse a dataset containing 2000 two-dimensional datapoints.

Typically, the number of MTPs computed by `SIA` is a very tiny fraction of all the patterns in a dataset. Moreover, many interesting repeated patterns in music are MTPs that can be discovered using `SIA` and `SIATEC`. Nevertheless, `SIATEC` typically discovers tens of thousands of TECs even in relatively short pieces such as those in Bach's *Das Wohltemperirte Klavier* and usually only a very small proportion of these TECs are perceptually significant. We are currently in the early stages of developing heuristics and algorithms for isolating those TECs that correspond to perceptually significant or analytically interesting repetitions. Our preliminary experiments suggest that, by using `SIATEC` in conjunction with carefully chosen heuristics, it may be possible to isolate various types of interesting repetition directly from multidimensional representations of musical scores or performances.

10 Future work

There are many ways in which the ideas reported here could be developed further. For example, the `SIA` and `SIATEC` algorithms could be used as the basis for new computational models and software tools for repeated pattern discovery, database indexing and compression. These models and tools could be used on data representing images, audio recordings, video, 3d-molecular models and, in general, anything that can appropriately be represented as a multidimensional dataset.

`SIA` and `SIATEC` currently only discover exactly repeated patterns in multidimensional datasets. Versions of these algorithms need to be developed that can discover approximately repeated patterns such as those that would occur, for example, in a MIDI file representation of an expressive performance of a musical work. This is not difficult to achieve. It involves modifying the notions of MTP and TEC. For example, the concept of an MTP defined in Eq.1 would have to be modified to:

$$MTP'(\mathbf{v}, D) = \{\mathbf{d_1} \mid \mathbf{d_1} \in D \land (\exists \mathbf{d_2} \mid \mathbf{d_2} \in D \land \mathbf{d_1} + \mathbf{v} \simeq \mathbf{d_2})\} \tag{22}$$

where '$\simeq$' denotes approximate equality.

In section 4 above we noted that multidimensional datasets could be used to represent both PCM audio data and time-varying audio frequency spectra. We also mentioned the possibility of using `SIATEC` to help with identifying instrumental parts in audio data. However, the algorithms described above would have to be modified in various ways before they could be used for processing audio data. For example, the algorithms would have to be capable of finding instances of approximate repetition. Also, the algorithms are too slow as they stand to process complete uncompressed audio files. It would therefore be necessary to modify them so that they can process large files in chunks or 'windows' (in which case the expected time complexity would become linear in the size of the file). Alternatively, feature extraction could be used to reduce the size of the audio file to be processed (see, for example, Kurth and Clausen, 2001).

We are also currently investigating the possibility of using `SIA` and `SIATEC` as the basis of a system for indexing music databases (Lemström *et al.*, 2001).

Algorithms and heuristics (possibly based on `SIA` and `SIATEC`) need to be developed for discovering and isolating various classes of perceptually significant repetition (e.g., theme-like repeated patterns, inversions, retrograde repetitions, augmentations, diminutions etc.).

Finally, there may be ways of improving the running times of `SIA` and `SIATEC`. In particular, it might be possible to achieve much faster running times by developing parallel versions of the algorithms or by exploiting techniques such as word-level parallelism (see, for example, Rahman and Raman, 1998).

Acknowledgements

David Meredith is supported by grant GR/N08049/01 from the EPSRC. Kjell Lemström was partially supported by grant number 48313 from the Academy of Finland and by grant GR/R25316 from the EPSRC. The authors would like to thank Marcus Pearce, Darrell Conklin and an anonymous reviewer for their helpful comments on earlier drafts of this paper.

References

Apostolico, A. and Ehrenfeucht, A. (1993). Efficient detection of quasiperiodicities in strings. *Theoretical Computer Science*, **119**(2), 247–265.

Apostolico, A. and Preparata, F. P. (1983). Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, **22**(3), 297–315.

Apostolico, A., Farach, M., and Iliopoulos, C. S. (1991). Optimal superprimitivity testing for strings. *Information Processing Letters*, **39**(1), 17–20.

Ben-Amram, A., Berkman, O., Iliopoulos, C. S., and Park, K. (1994). The subtree max gap problem with application to parallel string covering. In *Proceedings of the 5th ACM-SIAM Annual Symposium on Discrete Algorithms, Arlington, VA.*, pages 501–510.

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

Borowski, E. J. and Borwein, J. M. (1989). *Dictionary of Mathematics*. Collins.

Breslauer, D. (1992). An on-line string superprimitivity test. *Information Processing Letters*, **44**(6), 345–347.

Breslauer, D. (1994). Testing string superprimitivity in parallel. *Information Processing Letters*, **49**(5), 235–241. Available online at `http://citeseer.nj.nec.com/301674.html`.

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

Brinkman, A. R. (1990). *PASCAL Programming for Music Research*. The University of Chicago Press, Chicago and London.

Cambouropoulos, E. (1996). A general pitch interval representation: Theory and applications. *Journal of New Music Research*, **25**, 231–251.

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

Cambouropoulos, E., Crochemore, M., Iliopoulos, C. S., Mouchard, L., and Pinzon, Y. J. (1999). Algorithms for computing approximate repetitions in musical sequences. In R. Raman and J. Simpson, editors, *Proceedings of the 10th Australasian Workshop on Combinatorial Algorithms (AWOCA'99)*, pages 129–144, Perth, WA, Australia.

Cardon, A. and Crochemore, M. (1982). Partitioning a graph in $O(|A| \log_2 |V|)$. *Theoretical Computer Science*, **19**(1), 85–98.

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

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

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*. M.I.T. Press, Cambridge, Mass.

Crawford, T., Iliopoulos, C. S., and Raman, R. (1998). String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, **11**, 73–100.

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

Crochemore, M. and Rytter, W. (1994). *Text Algorithms*. Oxford University Press, Oxford.

Crow, D. and Smith, B. (1992). DB_Habits: Comparing minimal knowledge and knowledge-based approaches to pattern recognition in the domain of user-computer interactions. In R. Beale and J. Finlay, editors, *Neural Networks and Pattern Recognition in Human-Computer Interaction*, pages 39–63. Ellis Horwood, New York and London.

Dovey, M. (1999). An algorithm for locating polyphonic phrases within a polyphonic musical piece. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, pages 48–53, Edinburgh.

Dovey, M. and Crawford, T. (1999). Heuristic models of relevance ranking in searching polyphonic music. In *DIDEROT FORUM on Mathematics and Music: Computational and Mathematical Methods in Music*, pages 111–123.

Dovey, M. and Crawford, T. (2000). Heuristic models of relevance ranking in searching polyphonic music. Presented at the *London Strings Days 2000* workshop.

Floratos, A. and Rigoutsos, I. (2000). Method and apparatus for pattern discovery in 1-dimensional event streams. U.S. Patent no. 6,108,666.

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.

Huron, D. (1992). Design principles in computer-based music representation. In A. Marsden and A. Pople, editors, *Computer Representations and Models in Music*, pages 5–39. Academic Press, London.

Iliopoulos, C. S. and Mouchard, L. (1999). An $O(n \log n)$ algorithm for computing all maximal quasiperiodicities in strings. In *Proceedings of CATS'99: 'Computing: Australasian Theory Symposium'*, volume 213 of *Lecture Notes in Computer Science*, pages 262–272, Auckland, New Zealand. Springer-Verlag.

Iliopoulos, C. S. and Park, K. (1994). An optimal $O(\log \log n)$–time algorithm for parallel superprimitivy testing. *Journal of the Korean Information Science Society*, **21**(8), 1400–1404.

Iliopoulos, C. S. and Park, K. (1996). A work-time optimal algorithm for computing all string covers. *Theoretical Computer Science*, **164**(1–2), 299–310.

Iliopoulos, C. S., Moore, D. W., and Park, K. (1993). Covering a string. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching, Padova, Italy*, volume 684 of *Lecture Notes in Computer Science*, pages 54–62. Springer.

Kernighan, B. W. and Ritchie, D. M. (1988). *The C Programming Language*. Prentice-Hall International, London.

Krumhansl, C. L. (1990). *Cognitive Foundations of Musical Pitch*, volume 17 of *Oxford Psychology Series*. Oxford University Press, Oxford.

Kurth, F. and Clausen, M. (2001). Full-text indexing of very large audio data bases. In *Proceedings of the 110th AES Convention*, Amsterdam, The Netherlands.

Lemström, K. (2000). *String Matching Techniques for Music Retrieval*. Ph.D. thesis, University of Helsinki, Faculty of Science, Department of Computer Science. Report A-2000-4.

Lemström, K. and Tarhio, J. (2000). Searching monophonic patterns within polyphonic sources. In *Content-Based Multimedia Information Access Conference Proceedings (RIAO'2000), April 12–14, 2000*, volume 2, pages 1261–1279, Collége de France, Paris.

Lemström, K. and Ukkonen, E. (2000). Including interval encoding into edit distance based music comparison and retrieval. In *Proceedings of the AISB 2000 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science (April 17–18)*, pages 53–60, University of Birmingham.

Lemström, K., Haapaniemi, A., and Ukkonen, E. (1998). Retrieving music—to index or not to index. In *ACM Multimedia 98:—Art Demos—Technical Demos—Poster Papers—*, Bristol.

Lemström, K., Laine, P., and Perttu, S. (1999). Using relative interval slope in music information retrieval. In *Proceedings of the 1999 International Computer Music Conference*, pages 317–320, Beijing.

Lemström, K., Wiggins, G. A., and Meredith, D. (2001). A three-layer approach for music retrieval in large databases. In *Second Annual International Symposium on Music Information Retrieval (ISMIR 2001)*, Indiana University, Bloomington, Indiana.

Lerdahl, F. and Jackendoff, R. (1983). *A Generative Theory of Tonal Music*. M.I.T. Press, Cambridge, Mass. Cited by Krumhansl (1990).

Lewin, D. (1987). *Generalized Musical Intervals and Transformations*. Yale University Press, New Haven and London.

Li, Y. and Smyth, W. (1999). Computing the cover array in linear time. To appear.

Lincoln, H. (1967). Some criteria and techniques for developing computerized thematic indices. In Heckman, editor, *Electronische Datenverarbeitung in der Musikwissenschaft*. Regensburg.

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.pdf`.

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/public/mips20010910.pdf`.

Meredith, D., Lemström, K., and Wiggins, G. A. (2001). `SIA` and `SIATEC`: Efficient algorithms for translation-invariant pattern discovery in multi-dimensional datasets. Document submitted to UK Patent Office on May 23, 2001. Available online at `http://www.titanmusic.com/papers/public/patent.pdf`.

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

Moore, D. W. and Smyth, W. (1994). Computing the covers of a string in linear time. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 511–515.

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

Rahman, N. and Raman, R. (1998). An experimental study of word-level parallelism in some sorting algorithms. Available online at `http://citeseer.nj.nec.com/rahman98experimental.html`.

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. Cited by Krumhansl (1990).

Tanguiane, A. S. (1993). *Artificial Perception and Music Recognition*. Number 746 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin.

Ukkonen, E. (1985). Algorithms for approximate string matching. *Information and Control*, **64**, 100–118.

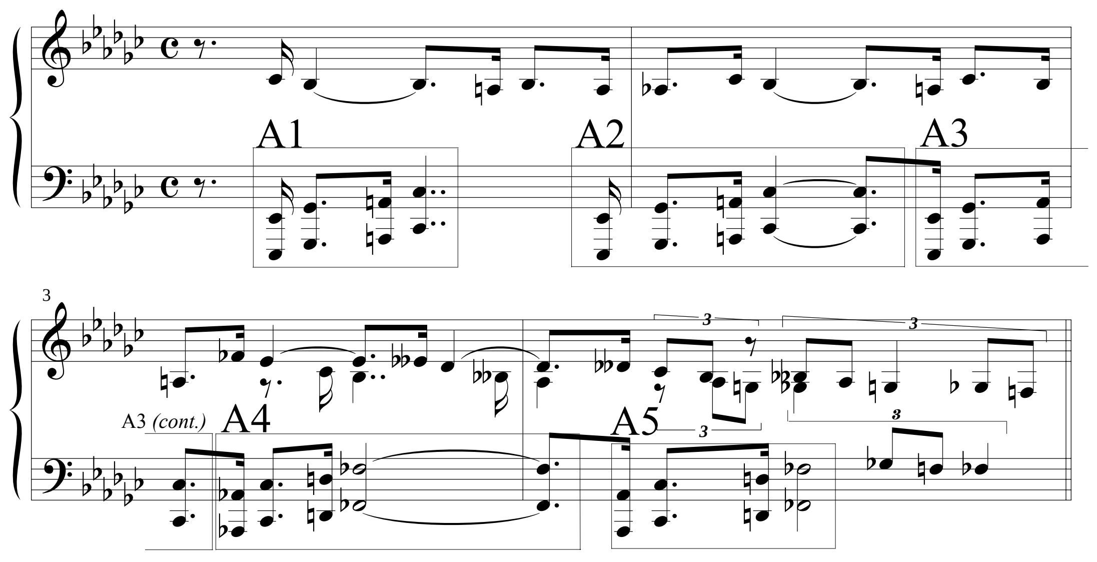

Figure 1: Measures 1–4 of Barber's Sonata for Piano, Op.26, showing 5 occurrences of the rising bass pattern A1–A5.

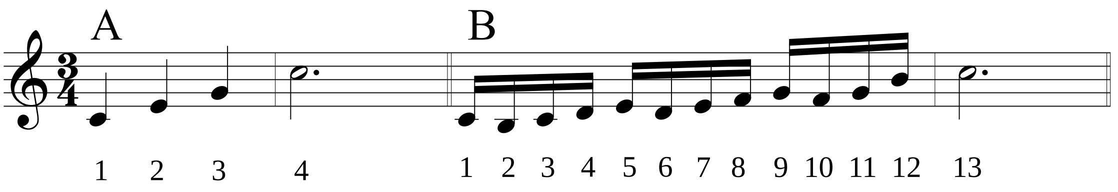

Figure 2: Example of a pattern being embellished using the technique of diminution.

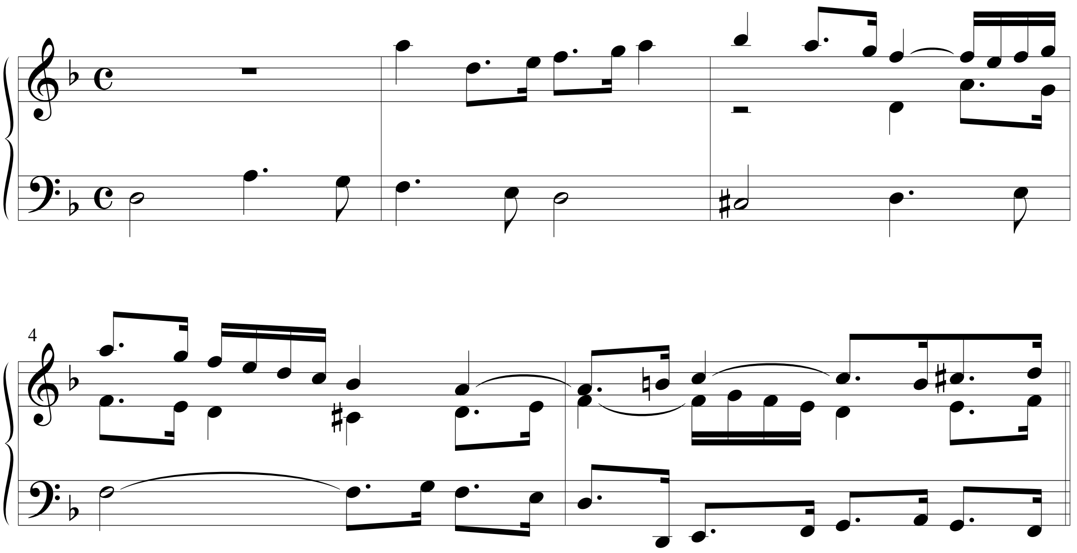

Figure 3: Measures 1–5 of *Contrapunctus VI* from *Die Kunst der Fuge* by J. S. Bach.

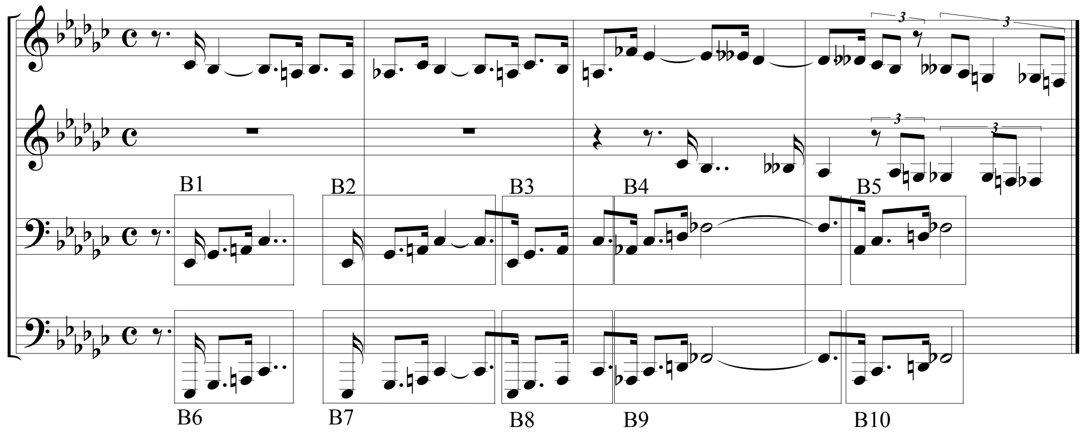

Figure 4: Measures 1–4 of Barber's Sonata for Piano, Op. 26, showing one natural way of partitioning the notes into monophonic voices.

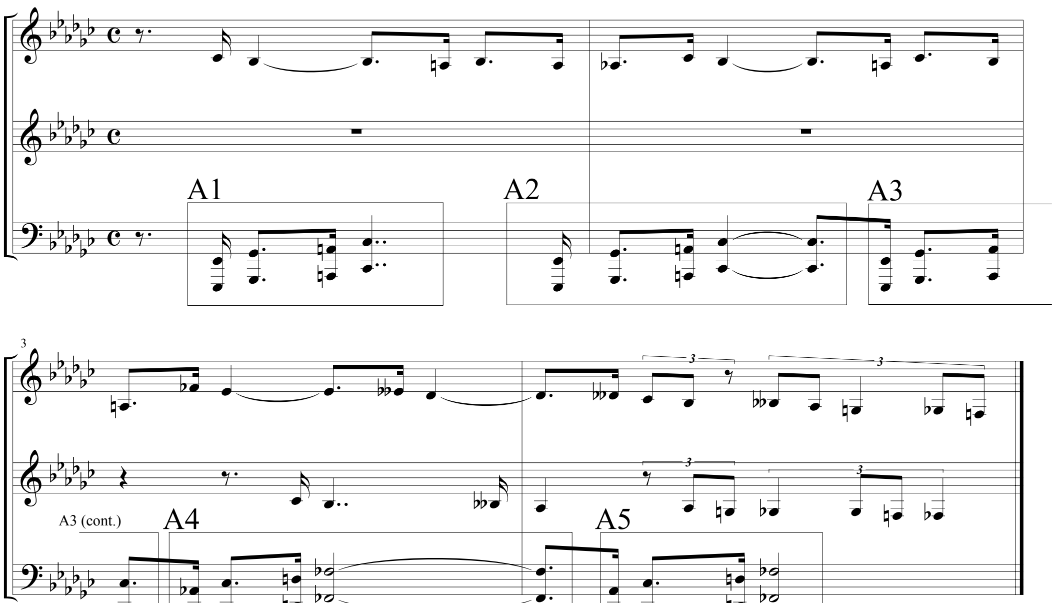

Figure 5: Measures 1–4 of Barber's Sonata for Piano, Op. 26, showing one natural way of partitioning the notes into polyphonic 'streams'.

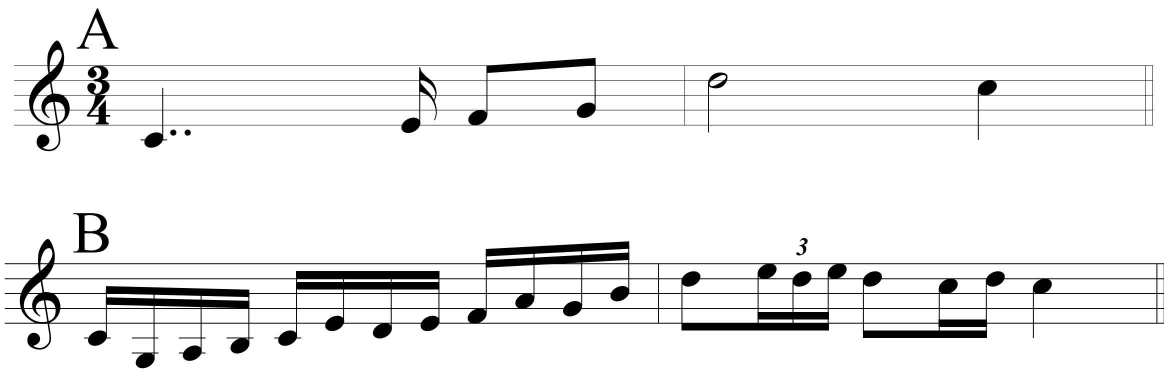

Figure 6: Pattern B is an embellished repetition of pattern A.

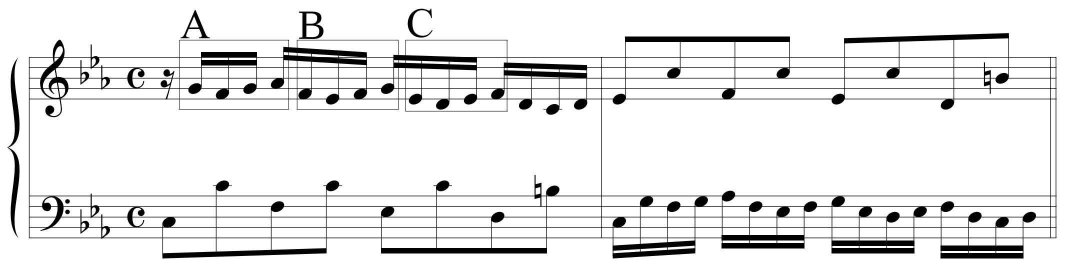

Figure 7: The first two measures of the Prelude in C minor from Book 2 of J. S. Bach's *Das Wohltemperirte Klavier*, BWV 871/1.

$$
\begin{array}{lllll}
\{ & \langle 0,27,16,2,2\rangle, & \langle 1,46,27,1,1\rangle, & \langle 2,39,23,2,2\rangle, & \\
 & \langle 2,44,26,1,1\rangle, & \langle 3,46,27,1,1\rangle, & \langle 4,32,19,2,2\rangle, & \\
 & \langle 4,47,28,1,1\rangle, & \langle 5,44,26,1,1\rangle, & \langle 6,39,23,2,2\rangle, & \\
 & \langle 6,42,25,1,1\rangle, & \langle 7,44,26,1,1\rangle, & \langle 8,30,18,2,2\rangle, & \\
 & \langle 8,46,27,1,1\rangle, & \langle 9,42,25,1,1\rangle, & \langle 10,39,23,2,2\rangle, & \\
 & \langle 10,41,24,1,1\rangle, & \langle 11,42,25,1,1\rangle, & \langle 12,29,17,2,2\rangle, & \\
 & \langle 12,44,26,1,1\rangle, & \langle 13,41,24,1,1\rangle, & \langle 14,38,22,2,2\rangle, & \\
 & \langle 14,39,23,1,1\rangle, & \langle 15,41,24,1,1\rangle, & \langle 16,27,16,1,2\rangle, & \\
 & \langle 16,42,25,2,1\rangle, & \langle 17,34,20,1,2\rangle, & \langle 18,32,19,1,2\rangle, & \\
 & \langle 18,51,30,2,1\rangle, & \langle 19,34,20,1,2\rangle, & \langle 20,35,21,1,2\rangle, & \\
 & \langle 20,44,26,2,1\rangle, & \langle 21,32,19,1,2\rangle, & \langle 22,30,18,1,2\rangle, & \\
 & \langle 22,51,30,2,1\rangle, & \langle 23,32,19,1,2\rangle, & \langle 24,34,20,1,2\rangle, & \\
 & \langle 24,42,25,2,1\rangle, & \langle 25,30,18,1,2\rangle, & \langle 26,29,17,1,2\rangle, & \\
 & \langle 26,51,30,2,1\rangle, & \langle 27,30,18,1,2\rangle, & \langle 28,32,19,1,2\rangle, & \\
 & \langle 28,41,24,2,1\rangle, & \langle 29,29,17,1,2\rangle, & \langle 30,27,16,1,2\rangle, & \\
 & \langle 30,50,29,2,1\rangle, & \langle 31,29,17,1,2\rangle & & \}
\end{array}
$$

Figure 8: A 5-dimensional dataset representation of the music in Figure 7.

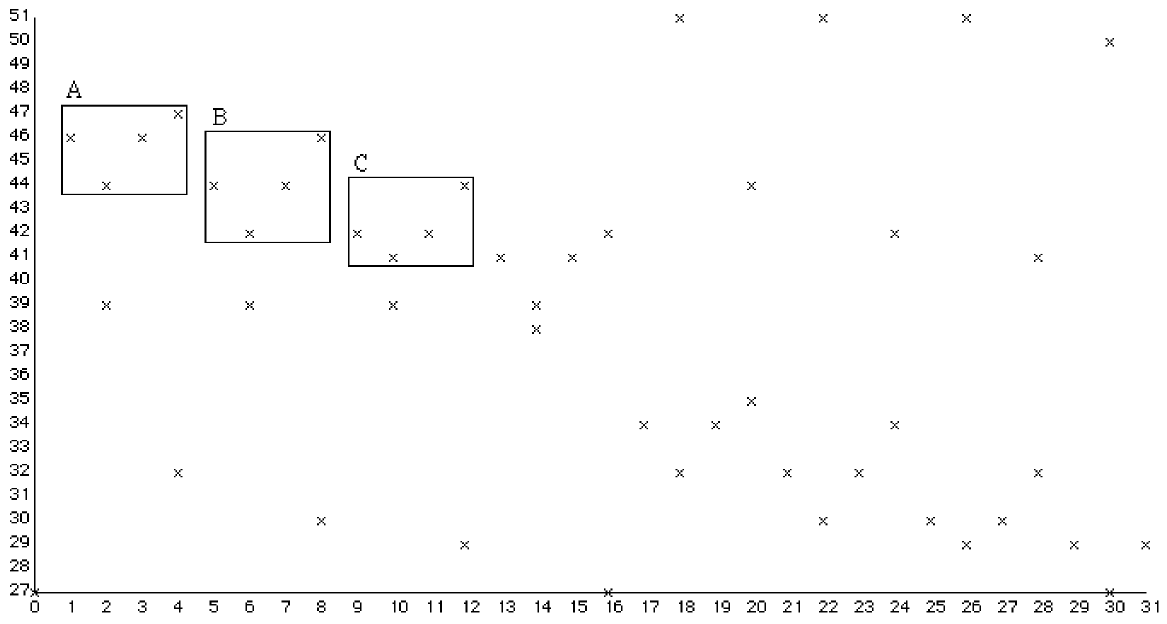

Figure 9: A two-dimensional orthogonal projection of the dataset in Figure 8 onto the plane defined by the onset time and chromatic pitch dimensions.

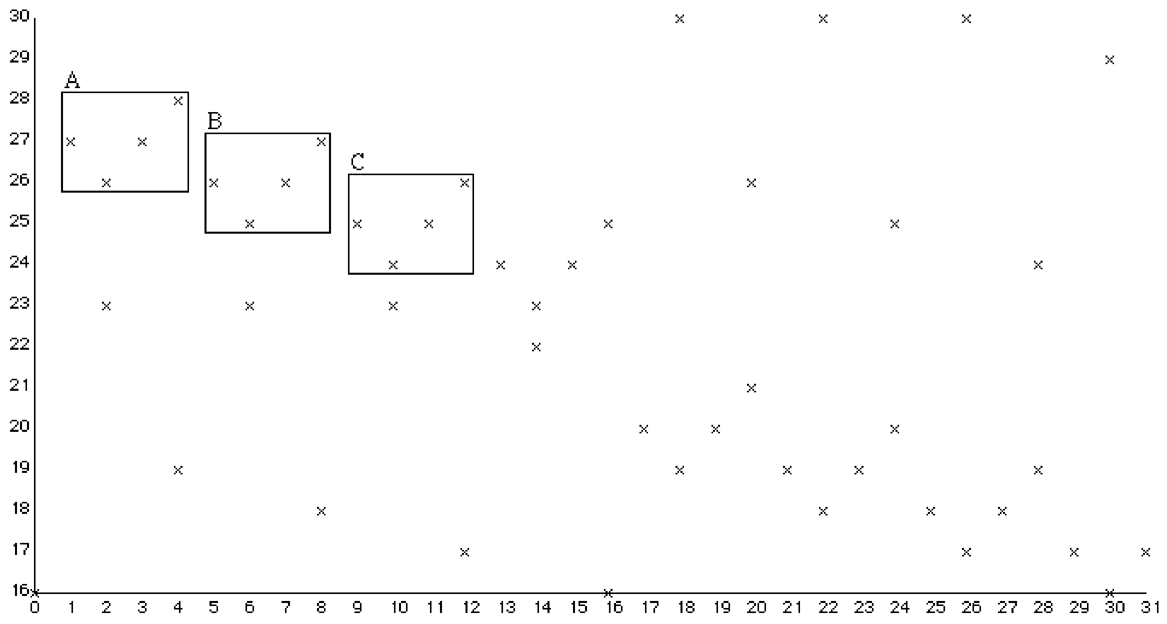

Figure 10: A two-dimensional orthogonal projection of the dataset in Figure 8 onto the plane defined by the onset time and morphetic pitch dimensions.

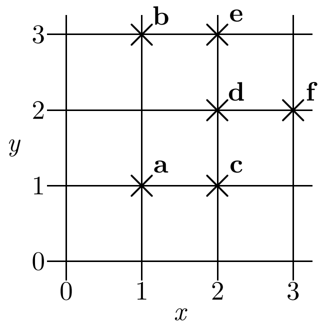

Figure 11: A simple two-dimensional dataset containing 6 datapoints.

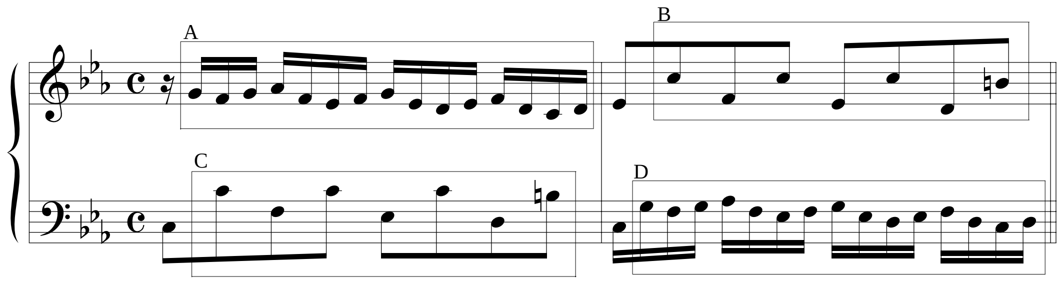

Figure 12: The first two measures of the Prelude in C minor from Book 2 of J. S. Bach's *Das Wohltemperirte Klavier*, BWV 871/1.

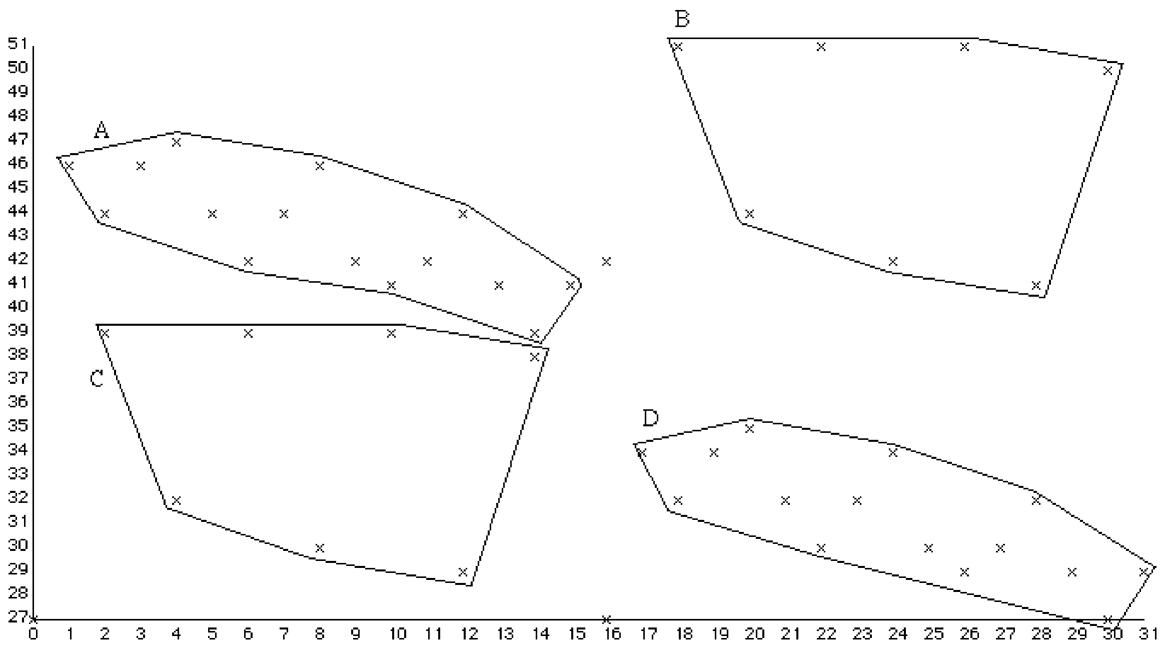

Figure 13: A two-dimensional orthogonal projection of the dataset in Figure 8 onto the plane defined by the onset time and chromatic pitch dimensions.

$$
\begin{array}{llllll}
\{ & \langle & \langle 0,1\rangle, & \{\langle 2,1\rangle,\langle 2,2\rangle\} & \rangle, & \\
 & \langle & \langle 0,2\rangle, & \{\langle 1,1\rangle,\langle 2,1\rangle\} & \rangle, & \\
 & \langle & \langle 1,-2\rangle, & \{\langle 1,3\rangle\} & \rangle, & \\
 & \langle & \langle 1,-1\rangle, & \{\langle 1,3\rangle,\langle 2,3\rangle\} & \rangle, & \\
 & \langle & \langle 1,0\rangle, & \{\langle 1,1\rangle,\langle 1,3\rangle,\langle 2,2\rangle\} & \rangle, & \\
 & \langle & \langle 1,1\rangle, & \{\langle 1,1\rangle,\langle 2,1\rangle\} & \rangle, & \\
 & \langle & \langle 1,2\rangle, & \{\langle 1,1\rangle\} & \rangle, & \\
 & \langle & \langle 2,-1\rangle, & \{\langle 1,3\rangle\} & \rangle, & \\
 & \langle & \langle 2,1\rangle, & \{\langle 1,1\rangle\} & \rangle & \}
\end{array}
$$

Figure 14: The set $\mathcal{S}(D)$ for the dataset in Figure 11.

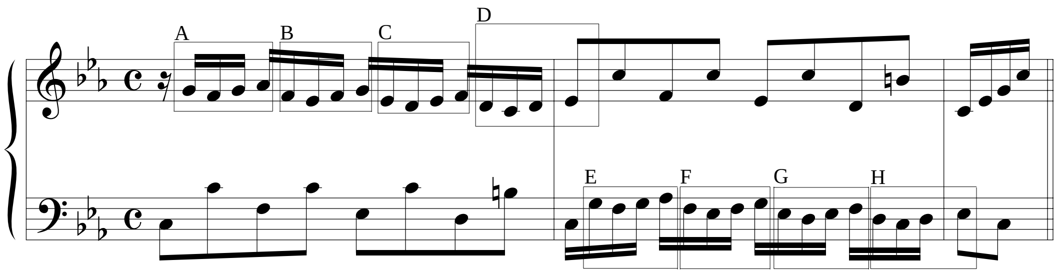

Figure 15: The opening measures of the Prelude in C minor from Book 2 of J. S. Bach's *Das Wohltemperirte Klavier*, BWV 871/1.

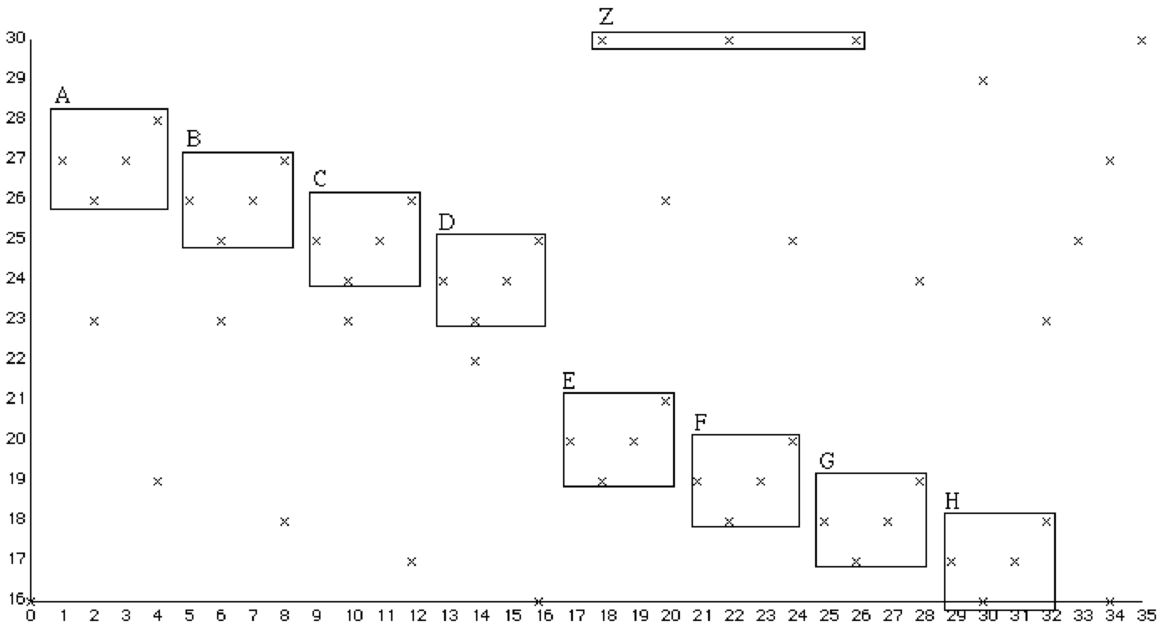

Figure 16: A two-dimensional dataset representation of the music in Figure 15 showing the onset time and morphetic pitch of each note.

$$
\begin{array}{lll}
\{ & \langle\{\langle 1,1\rangle,\langle 1,3\rangle,\langle 2,2\rangle\}, & \{\langle 0,0\rangle,\langle 1,0\rangle\}\rangle, \\
 & \langle\{\langle 1,1\rangle,\langle 2,1\rangle\}, & \{\langle 0,0\rangle,\langle 0,2\rangle,\langle 1,1\rangle\}\rangle, \\
 & \langle\{\langle 2,1\rangle,\langle 2,2\rangle\}, & \{\langle 0,0\rangle,\langle 0,1\rangle\}\rangle, \\
 & \langle\{\langle 1,1\rangle\}, & \{\langle 0,0\rangle,\langle 0,2\rangle,\langle 1,0\rangle,\langle 1,1\rangle,\langle 1,2\rangle,\langle 2,1\rangle\}\rangle \quad \}
\end{array}
$$

Figure 17: The set $\mathcal{T}'(D)$ for the dataset in Figure 11.

```
$m \leftarrow |\mathbf{V}|$
$i \leftarrow 1$
PRINT_NEW_LINE
PRINT('{')
while $i \leq m$
    PRINT('⟨')
    PRINT_VECTOR($\mathbf{V}[i,1]$)
    PRINT(',{')
    PRINT_VECTOR($\mathbf{D}[\mathbf{V}[i,2]]$)
    $j \leftarrow i+1$
    while $j \leq m$ and $\mathbf{V}[j,1] = \mathbf{V}[i,1]$
        PRINT(',')
        PRINT_VECTOR($\mathbf{D}[\mathbf{V}[j,2]]$)
        $j \leftarrow j+1$
    PRINT('}⟩')
    if $j \leq m$
        PRINT(',')
        PRINT_NEW_LINE
    $i \leftarrow j$
PRINT('}')
```

Figure 18: An algorithm for printing out $\mathcal{S}(D)$ using $\mathbf{V}$ and $\mathbf{D}$.

$$
\begin{aligned}
&\{\langle\langle 0,1\rangle,\{\langle 2,1\rangle,\langle 2,2\rangle\}\rangle,\\
&\langle\langle 0,2\rangle,\{\langle 1,1\rangle,\langle 2,1\rangle\}\rangle,\\
&\langle\langle 1,-2\rangle,\{\langle 1,3\rangle\}\rangle,\\
&\langle\langle 1,-1\rangle,\{\langle 1,3\rangle,\langle 2,3\rangle\}\rangle,\\
&\langle\langle 1,0\rangle,\{\langle 1,1\rangle,\langle 1,3\rangle,\langle 2,2\rangle\}\rangle,\\
&\langle\langle 1,1\rangle,\{\langle 1,1\rangle,\langle 2,1\rangle\}\rangle,\\
&\langle\langle 1,2\rangle,\{\langle 1,1\rangle\}\rangle,\\
&\langle\langle 2,-1\rangle,\{\langle 1,3\rangle\}\rangle,\\
&\langle\langle 2,1\rangle,\{\langle 1,1\rangle\}\rangle\}
\end{aligned}
$$

Figure 19: The output of the algorithm in Figure 18 for the dataset in Figure 11.

```
$m \leftarrow |\mathbf{V}|$
$i \leftarrow 1$
$\mathbf{X} \leftarrow \langle\,\rangle$
while $i \leq m$
    $\mathbf{Q} \leftarrow \langle\,\rangle$
    $j \leftarrow i + 1$
    while $j \leq m$ and $\mathbf{V}[j, 1] = \mathbf{V}[i, 1]$
        $\mathbf{Q} \leftarrow \mathbf{Q} \oplus \langle \mathbf{D}[\mathbf{V}[j, 2]] - \mathbf{D}[\mathbf{V}[j - 1, 2]] \rangle$
        $j \leftarrow j + 1$
    $\mathbf{X} \leftarrow \mathbf{X} \oplus \langle \langle i, \mathbf{Q} \rangle \rangle$
    $i \leftarrow j$
```

Figure 20: An algorithm for computing X using $\mathbf{V}$ and $\mathbf{D}$.

$$
\begin{array}{lllll}
\langle & \langle & 1, & \langle\langle 0,1\rangle\rangle & \rangle, \\
 & \langle & 3, & \langle\langle 1,0\rangle\rangle & \rangle, \\
 & \langle & 5, & \langle\,\rangle & \rangle, \\
 & \langle & 6, & \langle\langle 1,0\rangle\rangle & \rangle, \\
 & \langle & 8, & \langle\langle 0,2\rangle,\langle 1,-1\rangle\rangle & \rangle, \\
 & \langle & 11, & \langle\langle 1,0\rangle\rangle & \rangle, \\
 & \langle & 13, & \langle\,\rangle & \rangle, \\
 & \langle & 14, & \langle\,\rangle & \rangle, \\
 & \langle & 15, & \langle\,\rangle & \rangle \quad \rangle
\end{array}
$$

Figure 21: The ordered set $\mathbf{X}$ for the dataset in Figure 11.

$$
\begin{array}{llll}
\langle & \langle\ 5, & \langle\rangle & \rangle, \\
 & \langle\ 13, & \langle\rangle & \rangle, \\
 & \langle\ 14, & \langle\rangle & \rangle, \\
 & \langle\ 15, & \langle\rangle & \rangle, \\
 & \langle\ 1, & \langle\langle 0,1\rangle\rangle & \rangle, \\
 & \langle\ 3, & \langle\langle 1,0\rangle\rangle & \rangle, \\
 & \langle\ 6, & \langle\langle 1,0\rangle\rangle & \rangle, \\
 & \langle\ 11, & \langle\langle 1,0\rangle\rangle & \rangle, \\
 & \langle\ 8, & \langle\langle 0,2\rangle, \langle 1,-1\rangle\rangle & \rangle\quad\rangle
\end{array}
$$

Figure 22: The ordered set $\mathbf{Y}$ for the dataset in Figure 11.

```
$r \leftarrow |\mathbf{Y}|$
$m \leftarrow |\mathbf{V}|$
$i \leftarrow 1$
PRINT_NEW_LINE
PRINT('{')
if $r > 0$
    repeat
        $j \leftarrow \mathbf{Y}[i,1]$
        $\mathbf{I} \leftarrow \langle\rangle$
        while $j \leq m$ and $\mathbf{V}[j,1] = \mathbf{V}[\mathbf{Y}[i,1],1]$
            $\mathbf{I} \leftarrow \mathbf{I} \oplus \langle \mathbf{V}[j,2] \rangle$
            $j \leftarrow j+1$
        PRINT('⟨')
        PRINT_PATTERN($\mathbf{I}$)
        PRINT(',')
        PRINT_SET_OF_TRANSLATORS($\mathbf{I}$)
        PRINT('⟩')
        repeat
            $i \leftarrow i+1$
        until $i > r$ or $\mathbf{Y}[i,2] \neq \mathbf{Y}[i-1,2]$
        if $i \leq r$
            PRINT(',')
            PRINT_NEW_LINE
    until $i > r$
PRINT('}')
```

Figure 23: An algorithm for printing out $\mathcal{T}'(D)$.

```
PRINT_PATTERN($\mathbf{I}$)
1   $p \leftarrow |\mathbf{I}|$
2   PRINT('{')
3   PRINT_VECTOR($\mathbf{D}[\mathbf{I}[1]]$)
4   for $k \leftarrow 2$ to $p$
5       PRINT(',')
6       PRINT_VECTOR($\mathbf{D}[\mathbf{I}[k]]$)
7   PRINT('}')
```

Figure 24: The PRINT_PATTERN algorithm.

```
PRINT_SET_OF_TRANSLATORS($\mathbf{I}$)
$p \leftarrow |\mathbf{I}|$
$n \leftarrow |\mathbf{D}|$
if $p = 1$
   PRINT(‘{’)
   PRINT_VECTOR($\mathbf{W}[\mathbf{I}[1], 1, 1]$)
   for $k \leftarrow 2$ to $n$
      PRINT(‘,’)
      PRINT_VECTOR($\mathbf{W}[\mathbf{I}[1], k, 1]$)
   PRINT(‘}’)
else
   PRINT(‘{’)
   $\mathbf{J} \leftarrow \langle\rangle$
   for $k \leftarrow 1$ to $p$
      $\mathbf{J} \leftarrow \mathbf{J} \oplus \langle 1\rangle$
   $\mathit{FINISHED} \leftarrow \mathit{FALSE}$
   $\mathit{FIRST\_VECTOR} \leftarrow \mathit{TRUE}$
   $k \leftarrow 2$
   while not $\mathit{FINISHED}$
      if $\mathbf{J}[k] \leq \mathbf{J}[k-1]$
         $\mathbf{J}[k] \leftarrow \mathbf{J}[k-1] + 1$
      while $\mathbf{J}[k] \leq n - p + k$ and $\mathbf{W}[\mathbf{I}[k], \mathbf{J}[k], 1] < \mathbf{W}[\mathbf{I}[k-1], \mathbf{J}[k-1], 1]$
         $\mathbf{J}[k] \leftarrow \mathbf{J}[k] + 1$
      if $\mathbf{J}[k] > n - p + k$
         $\mathit{FINISHED} \leftarrow \mathit{TRUE}$
      else if $\mathbf{W}[\mathbf{I}[k], \mathbf{J}[k], 1] > \mathbf{W}[\mathbf{I}[k-1], \mathbf{J}[k-1], 1]$
         $k \leftarrow 2$
         $\mathbf{J}[1] \leftarrow \mathbf{J}[1] + 1$
         if $\mathbf{J}[1] > n - p + 1$
            $\mathit{FINISHED} \leftarrow \mathit{TRUE}$
      else if $k = p$
         if not $\mathit{FIRST\_VECTOR}$
            PRINT(‘,’)
         else
            $\mathit{FIRST\_VECTOR} \leftarrow \mathit{FALSE}$
         PRINT_VECTOR($\mathbf{W}[\mathbf{I}[k], \mathbf{J}[k], 1]$)
         $k \leftarrow 1$
         while $k \leq p$
            $\mathbf{J}[k] \leftarrow \mathbf{J}[k] + 1$
            if $\mathbf{J}[k] > n - p + k$
               $\mathit{FINISHED} \leftarrow \mathit{TRUE}$
               $k \leftarrow p$
            $k \leftarrow k + 1$
         $k \leftarrow 2$
      else
         $k \leftarrow k + 1$
   PRINT(‘}’)
```

Figure 25: The PRINT_SET_OF_TRANSLATORS algorithm.

$$\begin{aligned}
&\{\langle\{\langle 1,3\rangle\},\{\langle 0,-2\rangle,\langle 0,0\rangle,\langle 1,-2\rangle,\langle 1,-1\rangle,\langle 1,0\rangle,\langle 2,-1\rangle\}\rangle,\\
&\langle\{\langle 2,1\rangle,\langle 2,2\rangle\},\{\langle 0,0\rangle,\langle 0,1\rangle\}\rangle,\\
&\langle\{\langle 1,1\rangle,\langle 2,1\rangle\},\{\langle 0,0\rangle,\langle 0,2\rangle,\langle 1,1\rangle\}\rangle,\\
&\langle\{\langle 1,1\rangle,\langle 1,3\rangle,\langle 2,2\rangle\},\{\langle 0,0\rangle,\langle 1,0\rangle\}\rangle\}
\end{aligned}$$

Figure 26: The output of the algorithm in Figure 23 for the dataset in Figure 11.

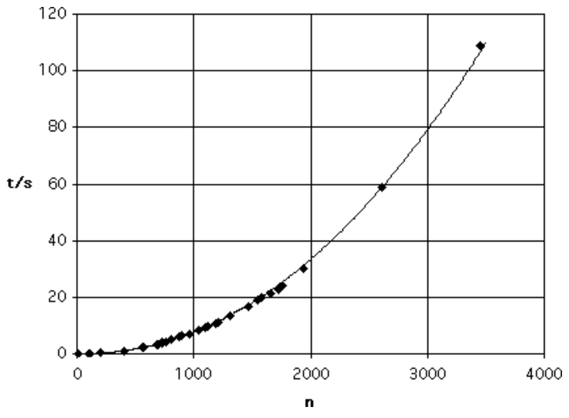

Figure 27: Graph of running time against dataset size for `SIA`.

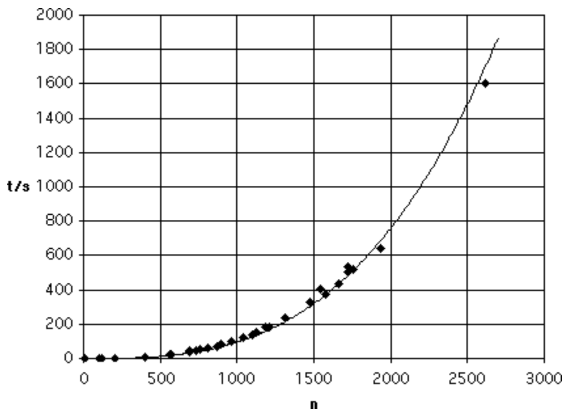

Figure 28: Graph of running time against dataset size for `SIATEC`.

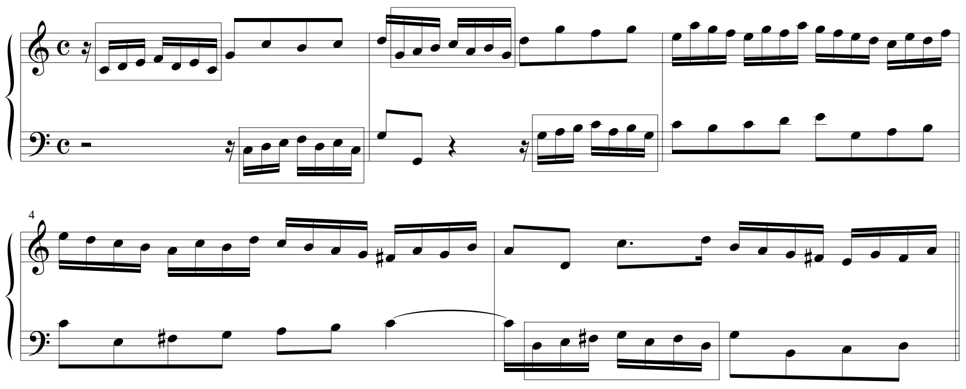

Figure 29: Bars 1–5 of Bach's two-part Invention in C major (BWV 772) showing one of the TECs discovered by `SIATEC`.

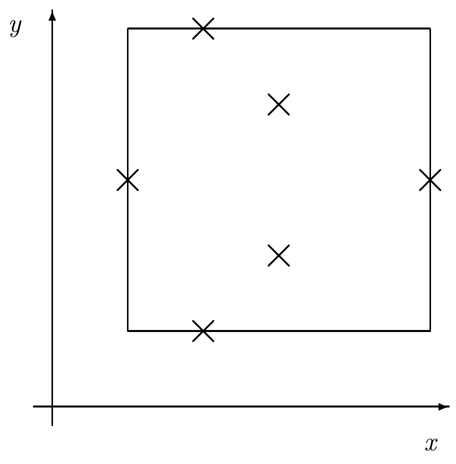

Figure 30: An example of a bounding box.

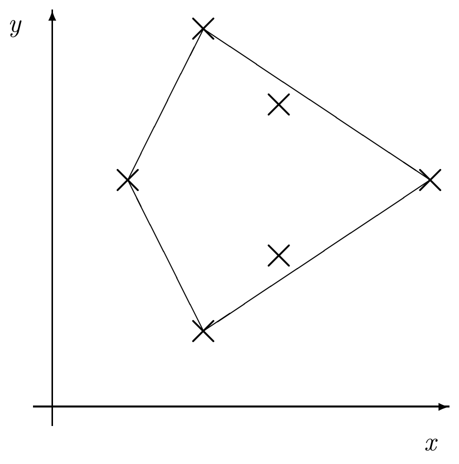

Figure 31: An example of a convex hull.

		From					
		$\langle 1,1\rangle$	$\langle 1,3\rangle$	$\langle 2,1\rangle$	$\langle 2,2\rangle$	$\langle 2,3\rangle$	$\langle 3,2\rangle$
	$\langle 1,1\rangle$						
	$\langle 1,3\rangle$	$\langle\langle 0,2\rangle,1\rangle$					
	$\langle 2,1\rangle$	$\langle\langle 1,0\rangle,1\rangle$	$\langle\langle 1,-2\rangle,2\rangle$				
To	$\langle 2,2\rangle$	$\langle\langle 1,1\rangle,1\rangle$	$\langle\langle 1,-1\rangle,2\rangle$	$\langle\langle 0,1\rangle,3\rangle$			
	$\langle 2,3\rangle$	$\langle\langle 1,2\rangle,1\rangle$	$\langle\langle 1,0\rangle,2\rangle$	$\langle\langle 0,2\rangle,3\rangle$	$\langle\langle 0,1\rangle,4\rangle$		
	$\langle 3,2\rangle$	$\langle\langle 2,1\rangle,1\rangle$	$\langle\langle 2,-1\rangle,2\rangle$	$\langle\langle 1,1\rangle,3\rangle$	$\langle\langle 1,0\rangle,4\rangle$	$\langle\langle 1,-1\rangle,5\rangle$	

Table 1: A vector table showing the set V for the dataset shown in Figure 11.

i	$\mathbf{V}[i]$	$\mathbf{D}[\mathbf{V}[i,2]]$	
1	$\langle\langle 0,1\rangle,3\rangle$	$\langle 2,1\rangle$	$=$ MTP for $\langle 0,1\rangle$
2	$\langle\langle 0,1\rangle,4\rangle$	$\langle 2,2\rangle$	
3	$\langle\langle 0,2\rangle,1\rangle$	$\langle 1,1\rangle$	$=$ MTP for $\langle 0,2\rangle$
4	$\langle\langle 0,2\rangle,3\rangle$	$\langle 2,1\rangle$	
5	$\langle\langle 1,-2\rangle,2\rangle$	$\langle 1,3\rangle$	$=$ MTP for $\langle 1,-2\rangle$
6	$\langle\langle 1,-1\rangle,2\rangle$	$\langle 1,3\rangle$	$=$ MTP for $\langle 1,-1\rangle$
7	$\langle\langle 1,-1\rangle,5\rangle$	$\langle 2,3\rangle$	
8	$\langle\langle 1,0\rangle,1\rangle$	$\langle 1,1\rangle$	
9	$\langle\langle 1,0\rangle,2\rangle$	$\langle 1,3\rangle$	$=$ MTP for $\langle 1,0\rangle$
10	$\langle\langle 1,0\rangle,4\rangle$	$\langle 2,2\rangle$	
11	$\langle\langle 1,1\rangle,1\rangle$	$\langle 1,1\rangle$	$=$ MTP for $\langle 1,1\rangle$
12	$\langle\langle 1,1\rangle,3\rangle$	$\langle 2,1\rangle$	
13	$\langle\langle 1,2\rangle,1\rangle$	$\langle 1,1\rangle$	$=$ MTP for $\langle 1,2\rangle$
14	$\langle\langle 2,-1\rangle,2\rangle$	$\langle 1,3\rangle$	$=$ MTP for $\langle 2,-1\rangle$
15	$\langle\langle 2,1\rangle,1\rangle$	$\langle 1,1\rangle$	$=$ MTP for $\langle 2,1\rangle$

Table 2: Reading the second column from top to bottom gives $\mathbf{V}$ for the dataset shown in Figure 11. The third column gives $\mathbf{D}[\mathbf{V}[i,2]]$ for each element $\mathbf{V}[i]$ in the second column. The right-hand side of the third column shows how the non-empty MTPs may be derived directly from $\mathbf{V}$.

		From					
		$\langle 1,1\rangle$	$\langle 1,3\rangle$	$\langle 2,1\rangle$	$\langle 2,2\rangle$	$\langle 2,3\rangle$	$\langle 3,2\rangle$
	$\langle 1,1\rangle$	$\langle\langle 0,0\rangle,1\rangle$	$\langle\langle 0,-2\rangle,2\rangle$	$\langle\langle -1,0\rangle,3\rangle$	$\langle\langle -1,-1\rangle,4\rangle$	$\langle\langle -1,-2\rangle,5\rangle$	$\langle\langle -2,-1\rangle,6\rangle$
	$\langle 1,3\rangle$	$\langle\langle 0,2\rangle,1\rangle$	$\langle\langle 0,0\rangle,2\rangle$	$\langle\langle -1,2\rangle,3\rangle$	$\langle\langle -1,1\rangle,4\rangle$	$\langle\langle -1,0\rangle,5\rangle$	$\langle\langle -2,1\rangle,6\rangle$
	$\langle 2,1\rangle$	$\langle\langle 1,0\rangle,1\rangle$	$\langle\langle 1,-2\rangle,2\rangle$	$\langle\langle 0,0\rangle,3\rangle$	$\langle\langle 0,-1\rangle,4\rangle$	$\langle\langle 0,-2\rangle,5\rangle$	$\langle\langle -1,-1\rangle,6\rangle$
To	$\langle 2,2\rangle$	$\langle\langle 1,1\rangle,1\rangle$	$\langle\langle 1,-1\rangle,2\rangle$	$\langle\langle 0,1\rangle,3\rangle$	$\langle\langle 0,0\rangle,4\rangle$	$\langle\langle 0,-1\rangle,5\rangle$	$\langle\langle -1,0\rangle,6\rangle$
	$\langle 2,3\rangle$	$\langle\langle 1,2\rangle,1\rangle$	$\langle\langle 1,0\rangle,2\rangle$	$\langle\langle 0,2\rangle,3\rangle$	$\langle\langle 0,1\rangle,4\rangle$	$\langle\langle 0,0\rangle,5\rangle$	$\langle\langle -1,1\rangle,6\rangle$
	$\langle 3,2\rangle$	$\langle\langle 2,1\rangle,1\rangle$	$\langle\langle 2,-1\rangle,2\rangle$	$\langle\langle 1,1\rangle,3\rangle$	$\langle\langle 1,0\rangle,4\rangle$	$\langle\langle 1,-1\rangle,5\rangle$	$\langle\langle 0,0\rangle,6\rangle$

Table 3: A vector table showing **W** for the dataset shown in Figure 11.