A Geometric Approach to Repetition Discovery and Pattern Matching in Polyphonic Music

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

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

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

Computer Science Colloquium
Department of Computer Science, King's College London

Wednesday, 21 November 2001.

1. A Geometric Approach to Repetition Discovery and Pattern Matching in Polyphonic Music

1. The diversity of perceptually significant repetition in music.
2. Most repetitions in music are not interesting.
3. Previous approaches to repetition discovery in music.
4. Representing music using multidimensional datasets.
5. `SIA`: Computing maximal repeated patterns.
6. `SIATEC`: Computing all the occurrences of each maximal repeated pattern.
7. Running `SIA` and `SIATEC` on music data.
8. Isolating perceptually significant repetitions.
9. `SIA(M)ESE`: Pattern matching in multidimensional datasets.
10. Some possible directions for further work.

1. A Geometric Approach to Repetition Discovery and Pattern Matching in Polyphonic Music

    1. [THANK COSTAS and STEFAN.]
    2. [E-MAIL LIST]
    3. I'm going to talk to you about the work that I've been doing with Geraint Wiggins and Kjell Lemström on repetition discovery and pattern matching in polyphonic music.
    4. Most of this talk will be about repetition discovery, and, in particular, the problem of developing an algorithm that can discover the perceptually significant repetitions in a passage of polyphonic music.
    5. I'll begin by presenting some examples of perceptually significant repetitions in music which will illustrate the fact that this class of phenomena is very diverse.
    6. I'll then show that, although the identification of perceptually significant repetitions is extremely important for achieving a rich understanding of a piece of music, typically, the vast majority of repetitions that occur within a piece are *not* interesting.
    7. It seems that most previous approaches to repetition discovery in music have been based on the assumption that the music to be analysed is represented in the form of a string or a set of strings.
    8. I'll briefly review a couple of these string-based approaches and I'll show that there seem to be certain types of perceptually significant repetition that occur quite often that are very difficult to find using a string-based approach.
    9. Also, it seems that if you want to find a wide range of types of perceptually significant repetition using a string-based approach, you generally have to run a variety of different algorithms on a number of different representations of the music.
    10. In our work we've avoided these difficulties by adopting a *geometric* approach in which the music to be analysed is represented as a multidimensional dataset—that is, a set of points in a Euclidean space.
    11. We've found that by doing this we're able to
        (a) process polyphonic music as easily and efficiently as monophonic music.
        (b) compute some of the repetitions that are difficult to find using a string-based approach.
        (c) dispense with multiple representations because we can simply run the same algorithms on various orthogonal projections of a single, rich multidimensional representation of the music.
    12. I'll present two repetition discovery algorithms, `SIA` and `SIATEC`, that are based on this approach.

(a) `SIA` computes all the maximal repeated patterns in a dataset.

(b) `SIATEC` computes all the occurrences of all the maximal repeated patterns in a dataset.

13. I'll then briefly talk about what happens when you run these algorithms on music data.

14. Our experiments suggest that the repeated patterns that we're interested in are often either equal to the maximal repeated patterns computed by `SIA` or derivable from them. However, typically, `SIA` also generates many patterns that are *not* musically interesting.

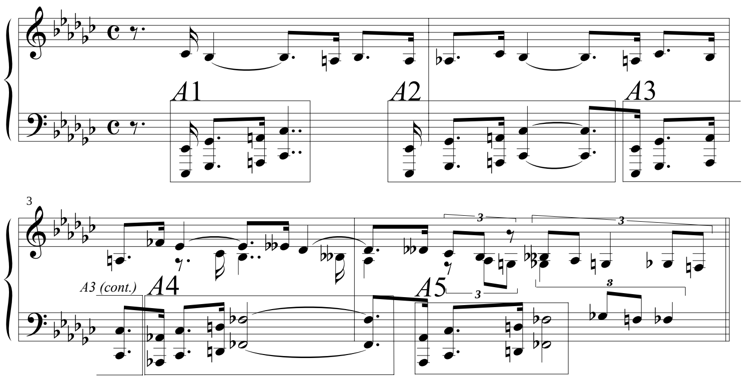
15. So some post-processing is therefore usually required to isolate the interesting repetitions in the output of `SIA` and `SIATEC` and I'll suggest a couple of heuristics that may be useful for doing this.

16. I'll then briefly describe a pattern-matching algorithm called `SIA(M)ESE`. This algorithm is based on `SIA` and it finds complete and partial matches of multidimensional query patterns in multidimensional datasets.

17. I should perhaps point out that although I'm going to be focusing on the musical applications of the algorithms, they are, in fact, quite general and could be used to process any data that can appropriately be represented in the form of a multidimensional dataset.

18. I'll finish up by suggesting some possible directions for further research.

2. The diversity of musical repetition



2. The diversity of musical repetition

1. Many music psychologists and music analysts have stressed that identifying the significant repetitions in a piece of music is an essential part of achieving a rich and satisfying interpretation of the piece.
2. Our work was originally motivated by the desire to develop a computational model of expert music cognition and it seems clear that one component of such a model would have to be able to discover perceptually significant repetitions.
3. However, the class of perceptually significant repetitions is a very diverse set. There are at least two reasons for this:
   (a) One reason is that the patterns involved in such repetitions vary widely in their structural characteristics.
   (b) A second reason is that there are many different ways of transforming a musical pattern to give another pattern that's perceived to be a version of it. For example, patterns can be truncated, augmented, diminished, inverted, reversed, embellished and so on.
4. I'll now present a couple of examples of significant repeated patterns that illustrate the diversity of this class of patterns.
5. A significant repeated pattern may be just a very small motif, consisting of no more than a few notes or it might be a whole section of a work containing hundreds of notes.
6. Here's an example of a very small perceptually significant repeated pattern from the beginning of Barber's Sonata for Piano, Op. 26. This example illustrates the general rule that for a very small pattern to be perceived as being significant, it generally has to be repeated many times—this one's repeated 5 times in the first 4 bars.
7. Here's what this example sounds like—try to listen to the rising bass pattern. [PLAY BARBER.MID.] Here's what it sounds like with the bass part emphasized. [PLAY BARBER MODIFIED.MID.]
8. On the other hand, in a sonata form movement it's typical for the whole exposition to be stated twice and then repeated in a modified form at the end of the movement. The exposition of a sonata-form movement often contains hundreds of notes. For example, there are Beethoven piano sonatas in which the exposition accounts for a quarter of all the notes in the first movement.
9. In polyphonic music in which the voices are unambiguously identifiable, the notes in a repeated pattern may all come from one voice or they may come from two or more voices. For example, in this *stretto* passage here taken from a Bach Fugue, each statement of the subject only contains notes from a single voice. [PLAY STRETTO.MID]

10. On the other hand, in this example from Mozart's G minor Symphony, each of the patterns involves the whole orchestra and contains notes from 13 voices. [PLAY MOZ.MID]

11. These two examples also show that the occurrences of a pattern may overlap as they do here [BACH] or they may occur consecutively, as they do here [Mozart] or they may be widely separated in the music as they often are, for example, in the case of the exposition and recapitulation of a sonata-form movement.

12. A significant repeated pattern may be what I call *temporally compact*—that is, it may contain all the notes that occur within the time interval spanned by the pattern as in this Mozart example here.

13. Alternatively, the pattern involved in a significant repetition might be *bounding box compact*—that is, it might contain all the notes in the piece that occur within the pattern's bounding box as in this Barber example.

14. This Bach extract illustrates yet another possibility where the repeated patterns are monophonic vocally compact patterns—that is, each pattern contains all the notes that occur in a single voice within the time period spanned by the pattern.

15. It's also possible to find examples of perceptually significant patterns that are not compact in any sense at all and I'll show you an example later on.

3. Most repetitions in music are not interesting

The pattern consisting of the notes in square boxes is an exact transposed repetition of the pattern consisting of the notes in elliptical boxes.

3. Most repetitions in music are not interesting

1. I'd now just like to demonstrate that although structurally significant repetitions are fundamental to a listener's understanding of piece of music, not all the repetitions that occur in a piece are interesting and significant.
2. For example, here we have the first few bars of Rachmaninoff's *Prelude* in C sharp minor, Op.3, No.2. The pattern consisting of the notes in round boxes is repeated 7 crotchets later, transposed up a minor ninth to give the pattern consisting of the notes in square boxes. [SHOW ON SLIDE.]
3. This is what these few bars sound like: [PLAY RACH-BS1-6.MID].
4. Now I'm going to play the same bars with the pattern notes emphasized: [PLAY BAD-PATTERN.MID].
5. Clearly, this repetition is just an artefact that results from the other musically significant repetitions that are occurring in this passage such as, for example, the exact repetition of bar 3 in bar 4.
6. In fact, it turns out that, typically, the vast majority of exact repetitions that occur within a piece of music are *not* musically interesting.
7. One of the main motivations behind our work is to develop algorithms that extract only the interesting repeated patterns of a particular type from the music.
8. Our task, therefore, involves formally characterising what it is about the interesting structural repetitions that distinguishes them from the many exact repetitions that the expert listener and analyst do not recognize as being structurally significant.

4. Previous approaches to repetition discovery in music

- Rolland 1999 (FlExPat)
  - Cannot be used for unvoiced polyphonic music.
  - Can only find patterns whose sizes lie within a user-specified range.
  - Too slow if allow patterns of any size.
  - Cannot find highly decorated repetitions.
- Hsu, Liu & Chen 1998
  - Cannot be used for unvoiced polyphonic music.
  - Cannot find transposed repetitions.
  - Slow (worst-case running time of $O(n^4)$).
  - Does not allow for gaps.
- Cambouropoulos 1998 (uses Crochemore 1981)
  - Does not allow for gaps.
  - Cannot be used for unvoiced polyphonic music.
- Conklin & Anagnostopoulou 2001
  - Allows crude repetition discovery at higher structural levels.
  - Only finds factors (not subsequences).

4. Previous approaches to repetition discovery in music

1. It seems that most previous attempts to develop a repetition discovery algorithm for music have been based on the assumption that the music to be analysed is represented as a string of symbols or a set of such strings.
2. An example of such an approach is Pierre-Yves Rolland's FlExPat program (Rolland 1999). This program can find approximate repetitions within a monophonic source. It could also be used to find repeated monophonic patterns in a polyphonic work in which each voice is represented as a string. Also, it is capable of finding repeated monophonic patterns which contain 'gaps'. However it suffers from a few weaknesses.
   (a) First, it cannot deal with unvoiced polyphonic music such as piano music.
   (b) Second, it can only find patterns whose sizes lie within a user-specified range and if the range is defined so that it allows patterns of any size, the overall worst-case running time goes up to at least $O(n^4)$.
   (c) Secondly, like most string-based approaches to approximate pattern matching, it uses the edit-distance approach. Unfortunately, such an approach is not typically capable of finding a repetition like the one shown here (CANT-FIND-THIS-2) because the edit distance between these two occurrences is actually quite large owing to the high number of insertions required to transform the plain version (A) into the ornamented one (B).
   (d) A program like Rolland's regards two patterns as being similar if the edit distance between them is less than some threshold k. However, for these two patterns to be considered 'similar' by Rolland's algorithm, this value of k would have to be set to at least 14 to allow for all these extra notes to be inserted. Unfortunately, this value would in general be too high because the program would then start regarding highly dissimilar patterns as being similar.
3. Hsu, Liu & Chen 1998 have also described a repetition discovery algorithm for music. Their algorithm is based on dynamic programming but it suffers from a number of serious weaknesses:
   (a) First, again, it cannot be used for analysing unvoiced polyphonic music.
   (b) Second, it is not capable, as described, of finding transposed repetitions.
   (c) Third, it has a worst-case running time of $O(n^4)$ which means it's too slow to be used for analysing large pieces.
   (d) It cannot find patterns 'with gaps'. That is, it can only find a pattern if it contains all the notes in the piece that occur during the time interval spanned by the pattern.
4. Cambouropoulos's (1998) *General Computational Theory of Musical Structure* also contains a pattern discovery component, that, in the most recent incarnation of the theory, is based on Crochemore's (1981) 'set partitioning' algorithm. In Cambouropoulos's theory, this pattern-discovery algorithm is used to help with determining the boundaries of the segments that are then categorised. Crochemore's algorithm

is very fast—it runs in $O(n \log_2 n)$ time. However, the algorithm does suffer from a few short-comings:

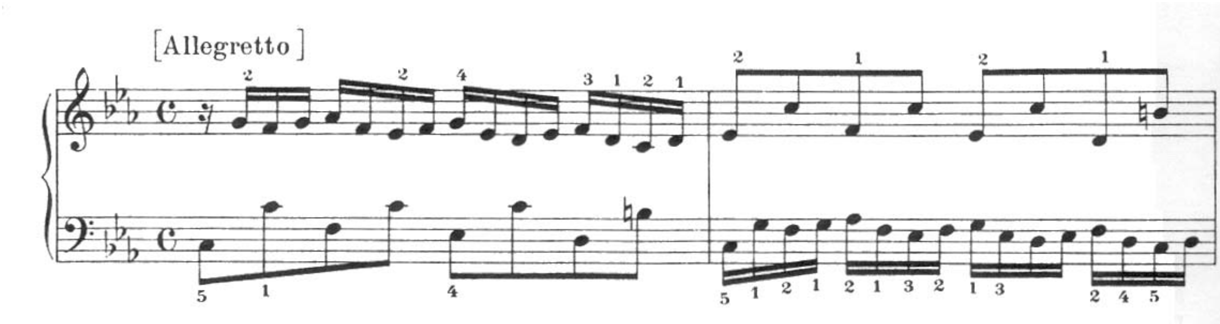
(a) It cannot find patterns with gaps.

(b) It cannot be used for finding patterns in unvoiced polyphonic music.

5. I'd also like to mention a recent approach described by Conklin & Anagnostopoulou 2001. In their method, a number of different string representations each representing a different 'viewpoint' on the music, are derived from a rich representation of the music to be analysed. They then discover repeated factors in these various string representations and isolate those factors that occur most frequently. Their approach is interesting because it offers a crude way of identifying repeated patterns at higher structural levels than the musical surface—one of their viewpoints, for example, represents just the first note in each crotchet beat. However, such an approach would not be capable of finding the example shown here because the notes that are common to both occurrences of the pattern do not all fall on the beat.

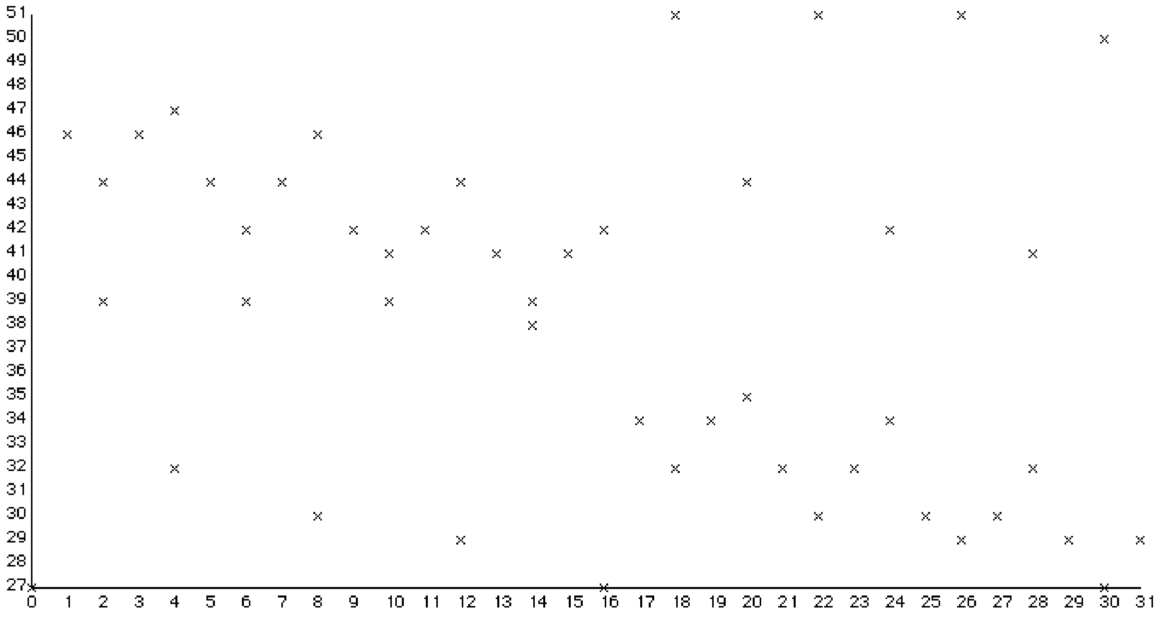
6. There is a multitude of string-processing algorithms available for discovering repeated factors in strings. However, there are far fewer algorithms available for finding repeated subsequences (i.e. patterns with gaps) and most of these seem to be NP-complete.

5. Representing music using multidimensional datasets (1)



$$\langle \text{onset time, chromatic pitch, morphetic pitch, duration, voice} \rangle$$

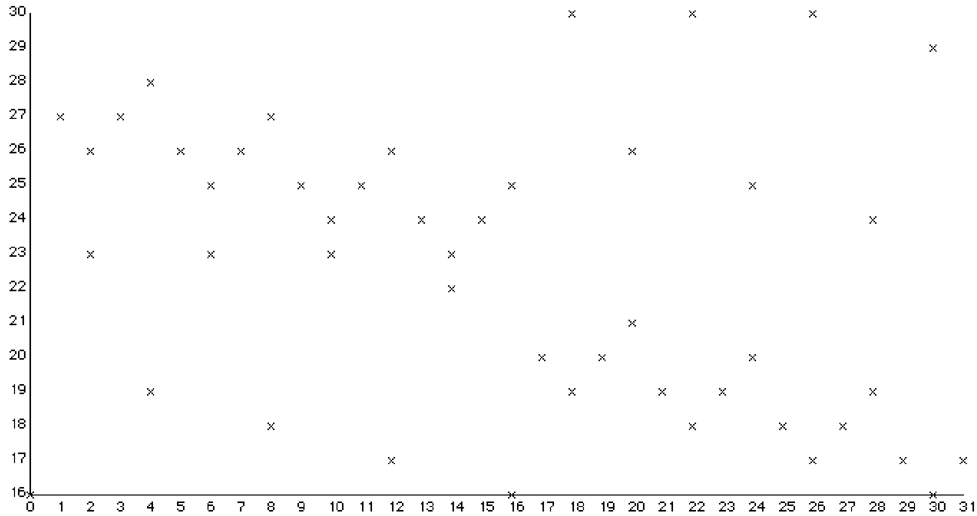
$$\begin{array}{llllll}
\{ & \langle 0,27,16,2,2\rangle, & \langle 1,46,27,1,1\rangle, & \langle 2,39,23,2,2\rangle, & \langle 2,44,26,1,1\rangle, & \langle 3,46,27,1,1\rangle, \\
 & \langle 4,32,19,2,2\rangle, & \langle 4,47,28,1,1\rangle, & \langle 5,44,26,1,1\rangle, & \langle 6,39,23,2,2\rangle,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

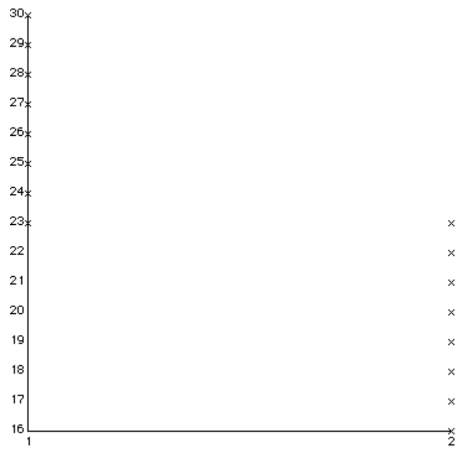


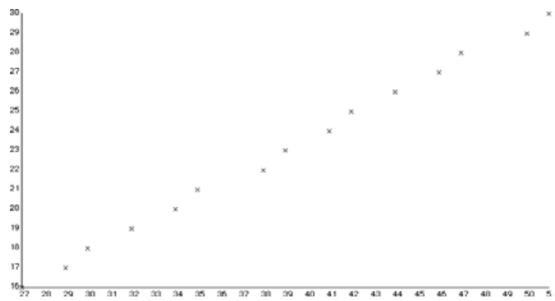
5. Representing music using multidimensional datasets (1)

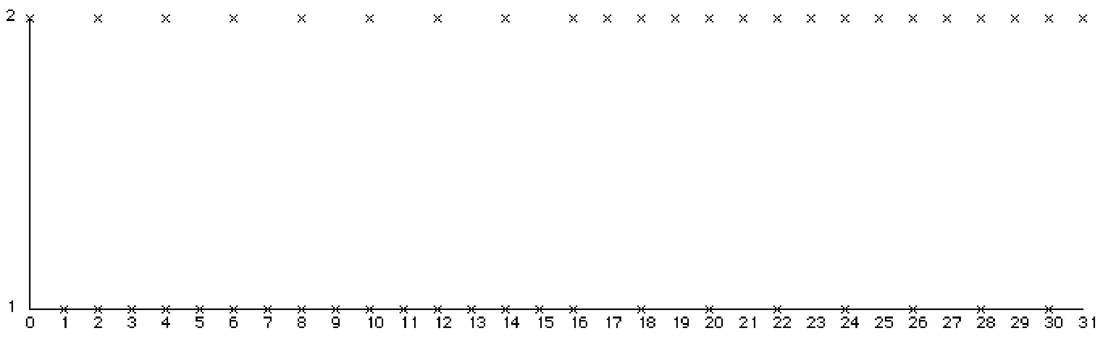
1. All the algorithms that I've just been talking about assume that the music to be analysed is represented either as a 1-dimensional string of symbols or, in the case of polyphonic music, as a set of such symbol strings.
2. And this assumption is the cause of many of their short-comings. For example, the fact that they process symbol strings means that these algorithms cannot deal with unvoiced polyphonic music such as keyboard music. Their string-matching basis also causes problems when it comes to finding patterns that are distributed between several voices or finding transposed occurrences of polyphonic patterns with gaps.
3. We've decided to avoid these problems by adopting a new, *geometric* approach in which the music to be analysed is represented as a multidimensional dataset.
4. A multidimensional dataset is simply a set of position vectors or datapoints in a Euclidean space with a finite number of dimensions. The algorithms that I'm going to describe work with datasets of any dimensionality and any size. Also the coordinates may take real values (which, of course, in an implementation would be represented as floating point values).
5. There are many possible appropriate ways of representing a piece of music as a multidimensional dataset and I'm going to use this example here to demonstrate some of the simpler possibilities.
6. At the top we have the first two bars of a Prelude from Bach's 48 Preludes and Fugues.
7. Underneath we have a 5-dimensional dataset that represents this score. The coordinate values in each datapoint represent onset time, chromatic pitch, morphetic pitch (which is continuous diatonic pitch), duration and voice. Each datapoint represents a single note event.
8. We can then consider various orthogonal projections of such a dataset. For example, we could just consider the first two dimensions and get this projection which tells us the chromatic pitch and onset time of each note.

6. Representing music using multidimensional datasets (2)





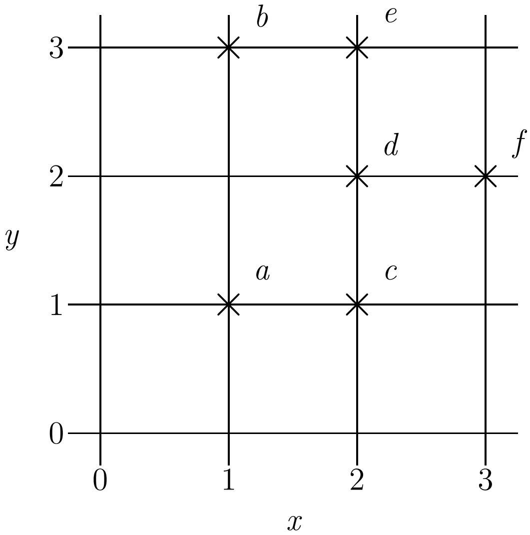




6. Representing music using multidimensional datasets (2)

1. Here are a number of other 2-dimensional projections of that dataset that give us useful information.
2. For example, this first one is a graph of morphetic pitch against onset time. Note that some of the patterns that were only similar in the chromatic pitch against onset time graph are now identical because we're using a representation of diatonic pitch. It's often more profitable when analysing tonal music to look for exact repetitions in this type of projection than in the chromatic pitch representation.
3. Here's another projection which shows pitch against voice and shows the range of each voice rather nicely.
4. This projection shows morphetic pitch against chromatic pitch and gives a representation of the pitch set that's used in the passage. In this particular case it shows quite clearly that the passage is in G major.
5. Finally, this projection shows voice against onset time and tells us the rhythm of each voice.
6. Adopting this geometric approach allows us to find classes of perceptually significant musical repetition that are very difficult to compute using string-based approaches.
7. It also allows us to process polyphonic music as simply and efficiently as monophonic music.
8. It dispenses with the need for multiple representations because we can run the same repetition discovery algorithms on various orthogonal projections of a single, rich multidimensional dataset representation.
9. It also allows us to discover repetitions in the dynamic, timbre and rhythmic structure of a piece as well as its pitch structure.

7. **`SIA`**: Discovering maximal repeated patterns in multidimensional datasets



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

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

7. `SIA`: Discovering maximal repeated patterns in multidimensional datasets

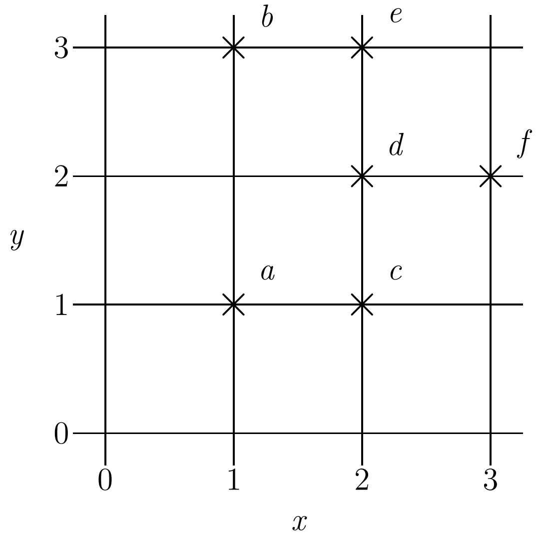
1. I'll now describe the `SIA` algorithm.
2. `SIA` takes a multidimensional dataset as input and finds for every possible vector the largest pattern in the dataset that can be translated by that vector to give another pattern in the dataset.
3. For example, if we consider this dataset here, then the largest pattern that can be translated by the vector $\langle 1, 0\rangle$ is the pattern $\{a, b, d\}$.
4. And the largest pattern that can be translated by the vector $\langle 1, 1\rangle$ is the pattern $\{a, c\}$.
5. We say that a pattern is *translatable* by a vector if it can be translated by the vector to give another pattern in the dataset.
6. And we say that the *maximal translatable pattern* or MTP for a vector is the largest pattern that can be translated by the vector to give another pattern in the dataset.
7. `SIA` discovers all the non-empty MTPs in a dataset and it does it like this:
8. First, the dataset is sorted.
9. Then the algorithm constructs this table here which we call the *vector table* for the dataset. A cell in the table contains the vector *from* the datapoint at the head of the column of that cell *to* the datapoint at the head of the row for that cell. [GIVE EXAMPLE.]
10. `SIA` computes all the values in this table below the leading diagonal as shown here. In other words, it computes for each datapoint all the vectors from that datapoint to every other datapoint in the dataset greater than it.
11. Note that each of these vectors is stored with a pointer that points back to the "origin" datapoint for which it was computed (that is, the datapoint at the top of its column).
12. Because the dataset has been sorted, the vectors increase as you descend the column and decrease as you move from left to right across a row.
13. Having constructed this table, `SIA` then simply sorts the vectors in the table using a slightly modified version of merge sort to give a list like this one here on the right-hand side.
14. Note that each vector in this list is still linked to the datapoint at the head of its column in the vector table. Simply reading off all the datapoints attached to the adjacent occurrences of a given vector in this list gives us the maximal translatable pattern for that vector.
15. The complete set of non-empty maximal translatable patterns can be obtained simply by scanning the list once, reading off the attached datapoints and starting a new pattern each time the vector changes. Each box in the right-hand column of the list corresponds to a maximal translatable pattern.

16. The most expensive step in this process is sorting the vectors which can be done in a worst-case running time of $O(kn^2 \log_2 n)$ for a k-dimensional dataset of size n.

17. The space complexity of the algorithm is $O(kn^2)$.

18. I understand that Costas's group has developed ways of sorting in a worst-case running time of $O(n \log_2 \log_2 n)$. If this could be applied to sorting vectors like this then that would mean that we could reduce the worst-case running time of `SIA` to $O(kn^2 \log \log n)$.

8. `SIATEC`: Discovering all the occurrences for each maximal translatable pattern



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

Time to find all occurrences of pattern of size $m = O(kmn)$.

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

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

Overall worst-case running time of `SIATEC` $= O(kn^3)$

8. `SIATEC`: Discovering all the occurrences for each maximal translatable pattern

1. I'll now describe our `SIATEC` algorithm.
2. `SIATEC` first generates all the maximal translatable patterns using a slightly modified version of `SIA` and then it finds all the occurrences of each MTP.
3. I explained on the previous slide that `SIA` only computes the vectors below the leading diagonal in the vector table. This is because the maximal translatable pattern for a vector $-v$ is the same as the pattern that you get by translating the maximal translatable pattern for v by the vector v itself. [DEMONSTRATE ON SLIDE.]
4. However, it turns out that by computing *all* the vectors in the vector table we can more efficiently discover all the occurrences of any given pattern within the dataset.
5. So in `SIATEC` we actually compute this complete table here and we use the region below the leading diagonal to compute the MTPs as in `SIA`.
6. We sort the dataset before computing the table so that the vectors increase as you descend a column and decrease as you move from left to right along a row.
7. Now, we know that a given column contains all the vectors that the datapoint at the top of the column can be translated by to give another point in the dataset.
8. Say we want to find all the occurrences of the pattern $\{a, c\}$ which is the maximal translatable pattern in this dataset for the vector $\langle 1, 1\rangle$.
9. Now, when we say that we want to "find all the occurrences" of a pattern, all we actually need to find is all the vectors that the pattern is translatable by. So, for example, the pattern $\{a, c\}$ is only translatable by the vectors $\langle 1, 1\rangle$ and $\langle 0, 2\rangle$.
10. We know that the column of vectors under a contains all the vectors that the point a can be translated by; and we know that the column under c contains all the vectors that the point c can be translated by. So we know that the pattern $\{a, c\}$ can only be translated by the vectors that occur in both of these columns.
11. In other words, to find the set of occurrences for a given pattern we simply have to find the intersection of the columns headed by the datapoints in the pattern.
12. By exploiting the orderedness of this table, we can find all the occurrences of a k-dimensional pattern of size m in a dataset of size n in a worst-case running time of $O(kmn)$.
13. We know that the complete set of maximal translatable patterns is found by `SIA` simply by sorting the vectors below the leading diagonal in the vector table. If there are l such patterns and m_i is the size of the ith pattern then this implies

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

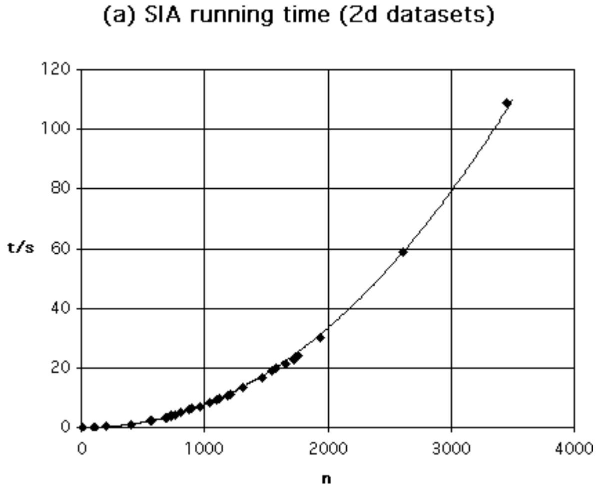
14. So the overall worst-case running time of `SIATEC` is

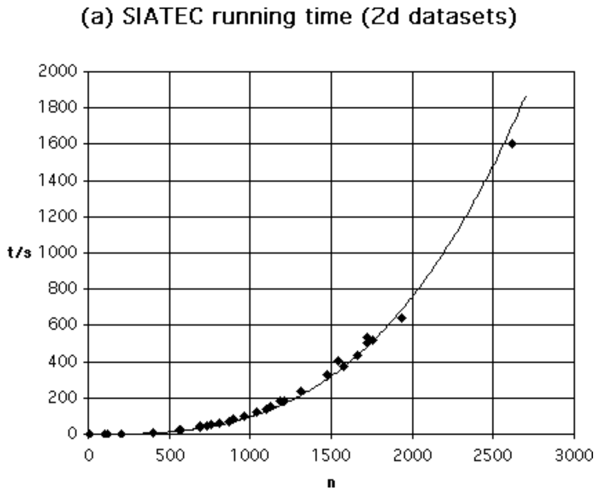
$$O\left(\sum_{i=1}^{l} k m_i n\right) \leq O\left(\frac{kn^2(n-1)}{2}\right)$$

So the algorithm is $O(kn^3)$ for a k-dimensional dataset of size n.

15. The space complexity is $O(kn^2)$.

9. Running SIA and SIATEC on music data.





- SIA and SIATEC implemented in C and run on 500MHz Sparc.
- Run on 52 datasets $6 \leq n \leq 3456$, $2 \leq k \leq 5$.
- 2 minutes for SIA to process piece containing 3500 notes.
- 13 minutes for SIATEC to process piece containing 2000 notes.

9. Running `SIA` and `SIATEC` on music data.

1. We've implemented `SIA` and `SIATEC` in C and run the programs on 52 datasets ranging in size from 6 to 3500 datapoints and in dimensionality from 2 to 5 dimensions.
2. We used a 500MHz Sparc machine.
3. The top graph here shows the running time of `SIA` on this machine for the 2-dimensional datasets in the sample. The smooth curve represents a running time of $kn^2 \log_2 n$.
4. The lower graph shows the running time of `SIATEC` on this machine for the 2-dimensional datasets in the sample. In this graph, the smooth curve represents a running time of kn^3.
5. As you can see, it took less than 2 minutes for `SIA` to process a piece containing 3500 notes and about 13 minutes for `SIATEC` to process a piece containing 2000 notes.

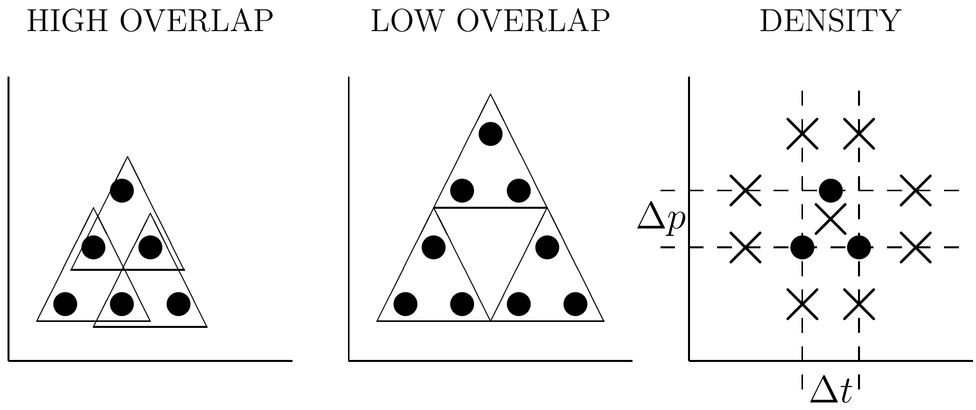
10. Isolating significant repetitions (1)

- Number of patterns in a dataset of size $n = 2^n$
- Number of patterns generated by `SIA` $< \frac{n^2}{2}$
- Experiments suggest that many interesting patterns are either equal to or derivable from the patterns generated by `SIA`.
- BUT many of the patterns generated by `SIA` are *not* musically interesting.
  - Over 70000 patterns discovered for Rachmaninoff *Prelude* Op.3 No.2.
  - Probably less than 100 of these are going to be analytically interesting.
- Need systems that evaluate the output of `SIATEC` and isolate various classes of musically significant repetitions.

10. Isolating significant repetitions (1)

1. A dataset of size n contains 2^n distinct subsets.
2. The number of patterns generated by `SIA` is less than $\frac{n^2}{2}$.
3. `SIA` discovers all the maximal translatable patterns in the powerset of a dataset and typically this set of maximal translatable patterns is only a tiny fraction of the patterns in the powerset of the dataset.
4. Our experiments suggest that the repeated patterns that we're interested in (including many that are very hard to find using string-matching techniques) are often either equal to the maximal translatable patterns generated by `SIA` or straightforwardly derivable from them.
5. Nevertheless, only a very small proportion of the patterns generated by `SIA` would be considered musically interesting by an analyst or expert listener. [SEE EXAMPLE ON SLIDE.]
6. This means that we need to devise systems that evaluate the output of `SIA` and `SIATEC` and isolate various classes of musically interesting repetitions.

11. Isolating significant repetitions (2)



OVERLAP = $\frac{3\times3}{6}$ = 1.5 OVERLAP = $\frac{3\times3}{9}$ = 1 TEMP. DENS = $\frac{3}{\Delta t}$

NOTE DENS. (TIME) = 3/8

NOTE DENS. (TIME & PITCH) = 3/4

Possible heuristics for finding "theme-like" patterns:

1. Frequency of occurrence.
2. Size of pattern.
3. Overlap.
4. Density
   (a) Temporal density.
   (b) Note density. "Region spanned by pattern" can be defined as time interval (segment) spanned by pattern, bounding-box, convex hull etc.

11. Isolating significant repetitions (2)

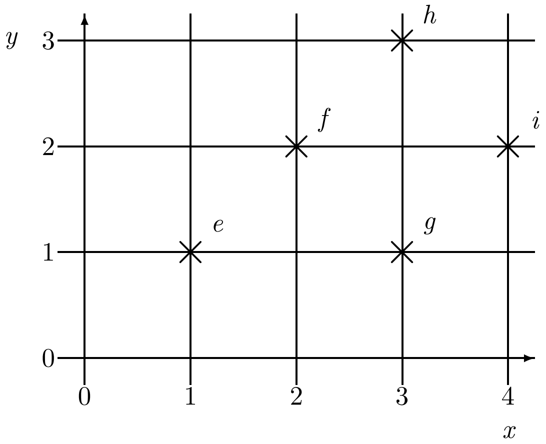
1. So, let's say we want to isolate 'theme-like' patterns—the sort of thing you might find in a musical thematic index like Barlow & Morgenstern 1983.
2. One approach would be to compute for each pattern a numerical value that's intended to reflect how "theme-like" the pattern is.
3. Here's a list of some of the heuristics that I've experimented with for doing this:
   (a) *FREQUENCY OF OCCURRENCE* The more frequently a pattern is repeated, the better.
   (b) *PATTERN SIZE* The larger a pattern, the better.
   (c) *OVERLAP* The fewer the number of notes shared between separate occurrences of the pattern, the better. This reflects the intuition that for a pattern to be perceived as being an individual unit, it must not share notes with repetitions of itself. [SEE EXAMPLE ON SLIDE.]
   (d) Cambouropoulos also uses these three quantities to select repeated patterns in his *GCTMS*.
   (e) *Density* The denser or more compact the pattern, the better. There are various ways in which one could define the density of a pattern:
      i. *Temporal density* Divide the number of notes in the pattern by the time interval spanned by it.
      ii. *Note density* Divide the number of notes in the pattern by the number of notes in region of the piece spanned by the pattern. There are a number of possible ways to define the "region spanned by a pattern". For example, this could be the time interval spanned by the pattern or the bounding box or convex hull of the pattern within its multidimensional representation.

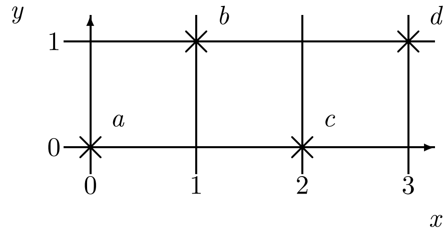
12. Some preliminary results

12. Some preliminary results

1. Although we've now run `SIA` and `SIATEC` on some quite large datasets representing complete pieces of music, we're still at the early stages of experimenting with heuristics to isolate perceptually significant patterns.
2. However, when I tried out the heuristics I've just described on some quite small datasets representing passages of music containing less than 200 notes, the results were quite encouraging.
3. For example, when I run `SIATEC` on the first five bars of this Bach Two-part Invention, 857 maximal translatable patterns are generated and the pattern that is judged to be the most "theme-like" using the heuristics I described on the previous slide is the seven-note subject of the Invention itself.
4. Here's what the passage sounds like with the occurrences of this pattern emphasized: [PLAY INVENTION-C.MID.]
5. [PUT ON SLIDE 2.]
6. When I ran the program on this passage from the beginning of Barber's Piano Sonata, this repeated bass figure is amongst those patterns that are evaluated to be the most "theme-like".
7. [PUT ON SLIDE 3.]
8. Similarly, when I run the program on the first 6 bars of the Rachmaninoff Prelude in C sharp minor this descending motif is one of the patterns judged to be most "theme-like".

13. `SIA(M)ESE`: Pattern matching in multidimensional datasets





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

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

13. `SIA(M)ESE`: Pattern matching in multidimensional datasets

1. As I mentioned at the beginning, we've also developed a pattern-matching algorithm based on `SIA` which we call `SIA(M)ESE`. I'll now briefly describe how this algorithm works.
2. `SIA(M)ESE` takes a multidimensional query pattern and a multidimensional dataset as input and finds all exact complete and partial matches of the query pattern in the dataset.
3. For example, let's imagine that we give this pattern and this dataset as input to `SIA(M)ESE` [SHOW ON SLIDE].
4. In this case, `SIA(M)ESE` will tell us, for instance, that the the complete query pattern $\{a, b, c, d\}$ can be matched to the pattern $\{e, f, g, i\}$ in the dataset. It will also tell us that the three point pattern $\{a, b, c\}$ in the query can be matched to the pattern $\{f, h, i\}$ in the dataset, that the points $\{c, d\}$ can be matched to $\{e, f\}$ and $\{f, h\}$ and so on.
5. It works in essentially the same way as `SIA`.
6. We begin by sorting the points in the query and the points in the dataset and then we construct a vector table like this one here.
7. Each entry in this table gives the vector *from* the query datapoint at the head of the colum in which the entry occurs *to* the dataset point at the head of the row in which the entry occurs. [GIVE EXAMPLE ON SLIDE].
8. Note that as in `SIA`, each vector in the vector table has a pointer that points back to the query pattern datapoint at the head of the column in which it occurs.
9. Having constructed this table, we then simply sort all the vectors in it to give a list like this one here.
10. This list gives us all the vectors that we can translate the query pattern by to give a non-empty match in the dataset.
11. The fact that each of these vectors still has a pointer to the query pattern datapoint at the head of its column in the vector table means that, for each vector, we can simply read off the points in the query pattern that have matches in the dataset when the query pattern is translated by that vector.
12. For example, if we look at the query pattern datapoints that are pointed to by vectors with the value $\langle 1, 1\rangle$ we find that all four of the points in the query pattern are matched which tells us that a complete occurrence of the query pattern occurs at a displacement of $\langle 1, 1\rangle$.
13. Similarly, if we look at the datapoints attached to the consecutive occurrences of the vector $\langle 2, 2\rangle$ in this list, we find that the points $\{a, b, c\}$ are matched when the query is translated by this vector.

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

14. Possible directions for further work

- Versions of `SIA` and `SIATEC` that discover *approximate* repetitions.
- Algorithms (possibly based on `SIA`) for discovering repetitions where the patterns are related by rotation, reflection or dilatation (enlargement).
- Improving running time of `SIA` and `SIATEC` by using word parallelism or by designing PRAM versions of the algorithms.
- Developing heuristics and algorithms for isolating various classes of perceptually significant repetition.
- Analysing in detail the difference between the types of repetition that can be discovered using `SIA` and those that can be discovered using string-based approaches.
- Developing applications:
  - `SIA` and `SIATEC`:
    - Data compression.
    - Database indexing.
    - Data mining.
  - `SIA(M)ESE`:
    - Information retrieval.
    - Computer-based learning systems.

Further information: `www.titanmusic.com`

Algorithms are the subject of a patent submitted on 23 May 2001.

14. Possible directions for further work

1. Finally, I'd like to suggest some possible directions for further work in this area. [USE SLIDE.]

Bibliography

Barlow, H. & Morgenstern, S. (1983), *A Dictionary of Musical Themes*, revised edn, Faber and Faber, London.

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

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

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

Hsu, J.-L., Liu, C.-C. & 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', Association of Computing Machinery, pp. 281–288.

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