Pattern Discovery and Matching in Polyphonic Music and Other Multidimensional Datasets

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

Fifth World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2001)
Sheraton World Resort, Orlando, FL., U.S.A.

Tuesday, 24 July 2001.

1. Pattern Discovery and Matching in Polyphonic Music and Other Multidimensional Datasets

1. The variety of musically significant repeated patterns.
2. Not all repetition is significant.
3. Other approaches to pattern discovery in music.
4. `SIA`: Finding maximal repeated patterns.
5. `SIATEC`: Finding all the occurrences of each maximal repeated pattern.
6. Using heuristics to select the patterns we're interested in.
7. Some preliminary results.
8. `SIA(M)ESE`: Flexible pattern matching in multidimensional datasets.

1. Pattern Discovery and Matching in Polyphonic Music and Other Multidimensional Datasets

1. We're going to be talking to you about pattern discovery and matching in polyphonic music and in multidimensional datasets in general.

2. I'll talk for about 20 minutes about machine *discovery* of repeated structures in music, focusing on `SIA` and `SIATEC`, which are two new pattern-discovery algorithms that I've developed over the past year in collaboration with Geraint Wiggins and Kjell Lemström.

3. Geraint will then explain how `SIA` can be adapted and generalised to produce an efficient and flexible pattern-*matching* algorithm which he calls `SIA(M)ESE`.

4. We're both going to concentrate on the musical applications of these algorithms but you should be aware that these algorithms are, in fact, quite general and could be applied to any data that can appropriately be represented in the form of a multidimensional dataset (that is, a set of points in a Cartesian space.)

5. I'll begin by presenting some examples of significant repetition in music which will illustrate the fact that the class of musically interesting repeated patterns is a very diverse set containing patterns with widely different structural characteristics.

6. I'll then show that, although musically significant repetitions are extremely important for understanding a piece of music, not all the structural repetitions that occur in a piece are significant.

7. I'll then briefly review some other approaches to pattern discovery in music and I'll show how some of their shortcomings can be overcome by abandoning the use of symbol strings to represent music in favour of a multidimensional representation.

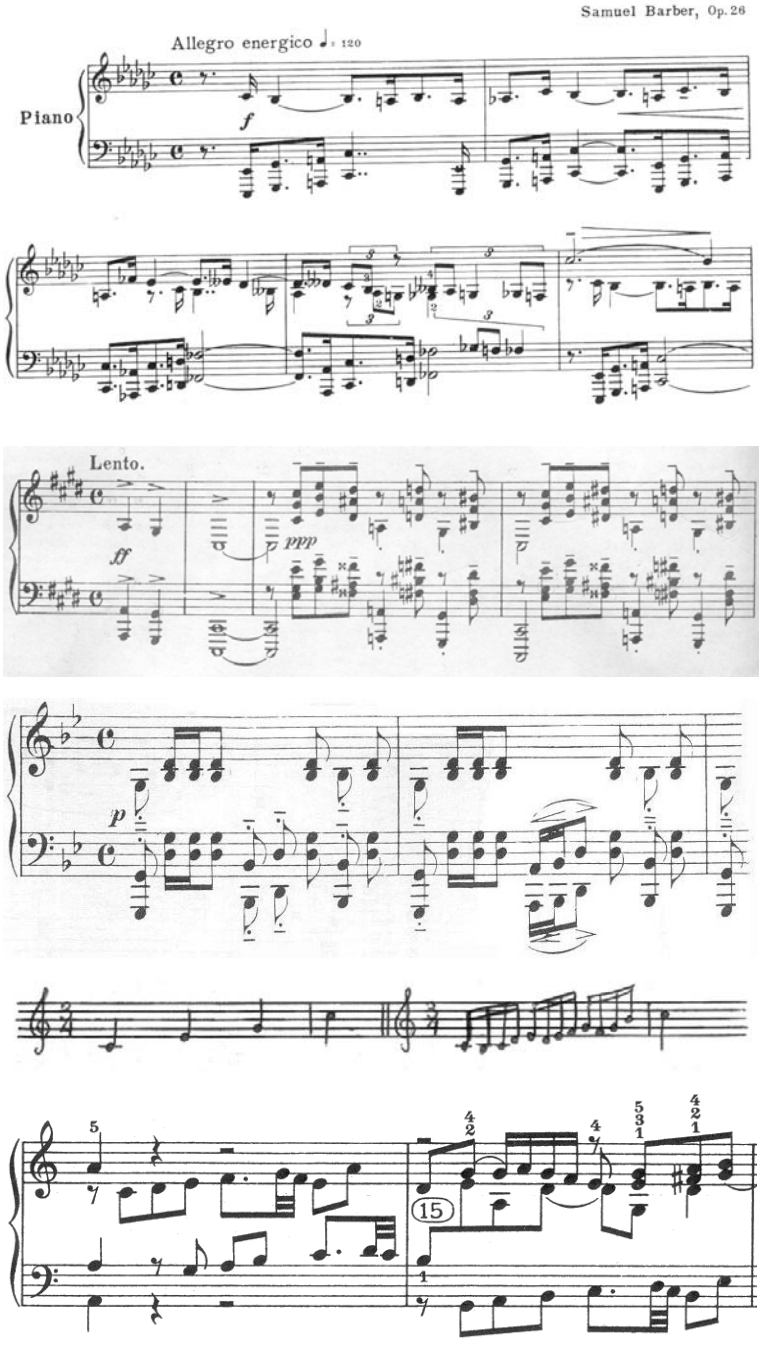
8. I'll then present my pattern discovery algorithms, `SIA` and `SIATEC`. `SIA` finds two occurrences of each maximal repeated pattern in a dataset; and `SIATEC` finds *all* the occurrences of each maximal repeated pattern.

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

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

11. I'll then hand over to Geraint who will describe his pattern-matching algorithm `SIA(M)ESE`.

2. The variety of repeated patterns in music



2. The variety of repeated patterns in music

1. Many music psychologists and music analysts have stressed that identifying the significant instances of repetition in a piece of music is an essential step in the process by which a listener achieves a rich and satisfying understanding 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 the 'perceptually significant' repeated patterns in a musical surface.

3. However, the class of 'perceptually significant' repeated patterns contains patterns with very diverse structural characteristics.

4. For example, a musically significant repeated pattern may be just a very small motif, consisting of no more than a few notes. Usually, for such patterns to be perceived as structural units they have to occur frequently over the course of a piece. This happens, for example, in much of Brahms's so-called 'monothematic' music. It also occurs here in the opening bars of Barber's *Sonata for Piano* where the left-hand motif is repeated over and over again. Here's what it sounds like: [PLAY BARBER.MID.] And here's what it sounds like with that repeated motif emphasized: [PLAY BARBERMODIFIED.MID.]

5. On the other hand, in a sonata form movement there may be large chunks of the exposition that are repeated in the recapitulation and each repeated chunk may consist of hundreds of notes.

6. So significant repeated patterns may be very small or very large.

7. They may also be either polyphonic or monophonic. In many simple songs, the significant repeated patterns are often purely monophonic. However, in polyphonic music and music like piano music where the voicing is not explicitly represented, a significant repeated pattern may consist of overlapping notes from several different voices. Here's a simple example of this from the beginning of Rachmaninoff's famous C sharp minor *Prelude*: the whole of bar 4 is a repeat of bar 3.

8. A significant repeated pattern may be what I call a *compact segment* that contains all the notes that occur within a particular time interval during the piece—as in bars 3 and 4 here—or it may contain only some of the notes that occur during the time interval that it spans—as in this Barber example.

9. If the pattern contains only some of the notes that occur during the time interval that it spans then usually the pattern contains all the notes in one of the voices or all the notes in two or more of the voices (as it sort of does here in the Barber example).

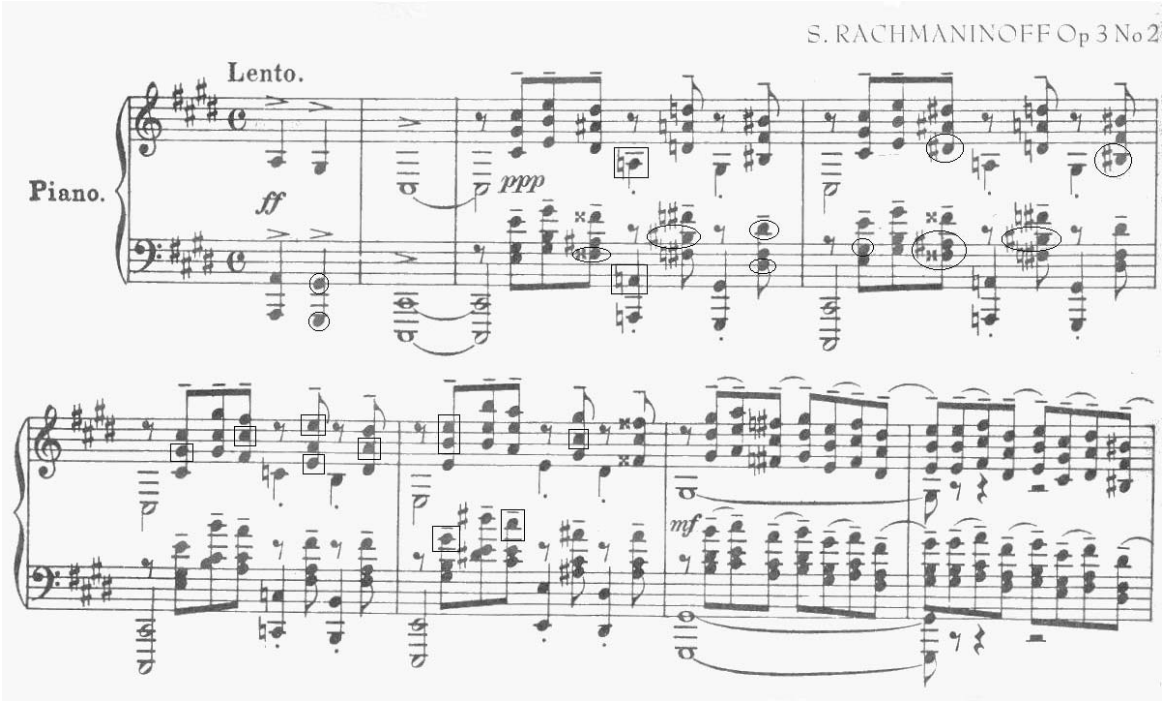
10. However, in keyboard music where the voicing is often not explicitly notated and where the structural voices are not distinguished from eachother by timbre, it might not be appropriate to represent the music as being composed of well-defined voices. It may be that all we can definitely say is that a repeated pattern contains some but not all of the notes that occur in the time interval spanned by the pattern.

11. Sometimes, in the case of unvoiced polyphonic music like keyboard music, a significant repeated pattern may exclude not only some of the notes that occur within the time interval spanned by the pattern but also some of the notes in the pitch-range spanned by the pattern, as in this example from another of Rachmaninoff's *Preludes*, this time Op.23, No.5. This is what it sounds like: [PLAY PREL235.MID]. And here's what it sounds like with the repeated pattern emphasized: [PLAY PREL235EMPH.MID].

12. A pattern can also be repeated in an ornamented form. For example, here we have a rising C major arpeggio that's elaborated using a technique known as 'diminution' which involves inserting shorter ornamental notes in between the main notes of the pattern. This sounds like this: [PLAY ARPEGGIO.MID].

13. Or occurrences of the pattern may overlap in time as occurs when you get a *stretto* in a contrapuntal piece as shown here in this extract from a fugue by Bach which sounds like this: [PLAY STRETTO.MID].

3. Not all repetition is musically significant



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. Not all repetition is musically significant

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. Other pattern-discovery algorithms for 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.

4. Other pattern-discovery algorithms for music

1. It seems that most previous attempts to develop a pattern discovery algorithm for music have been based on string-matching techniques.

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. However it suffers from a few weaknesses.

   (a) First, it cannot deal with unvoiced polyphonic music because it is based on the assumption that the music is represented using one-dimensional symbol strings.

   (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) Finally, 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 (ARPEGGIO) 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 into the ornamented one.

   (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 9. 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 pattern 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) And finally, 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.

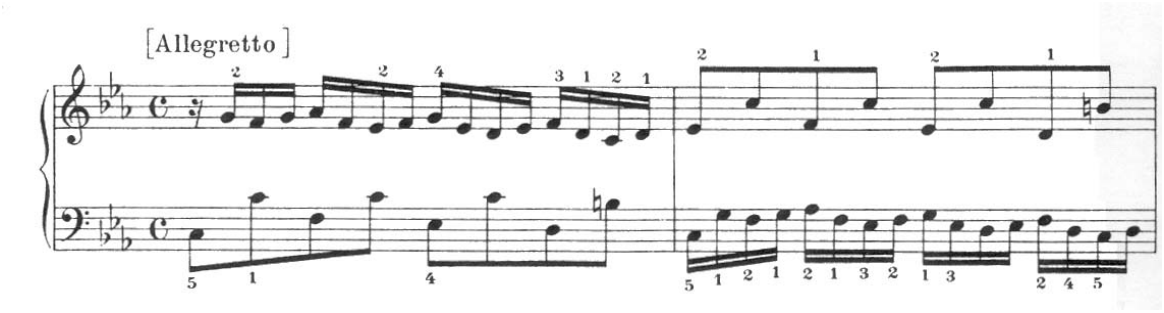
4. Cambouropoulos's (1998) *General Computational Theory of Musical Structure* also contains a pattern discovery component, that, in the most recent incarnation of the theory, is based on Crochemore's (1981) 'set partitioning' algorithm. In Cambouropoulos's theory, this pattern-discovery algorithm is 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.

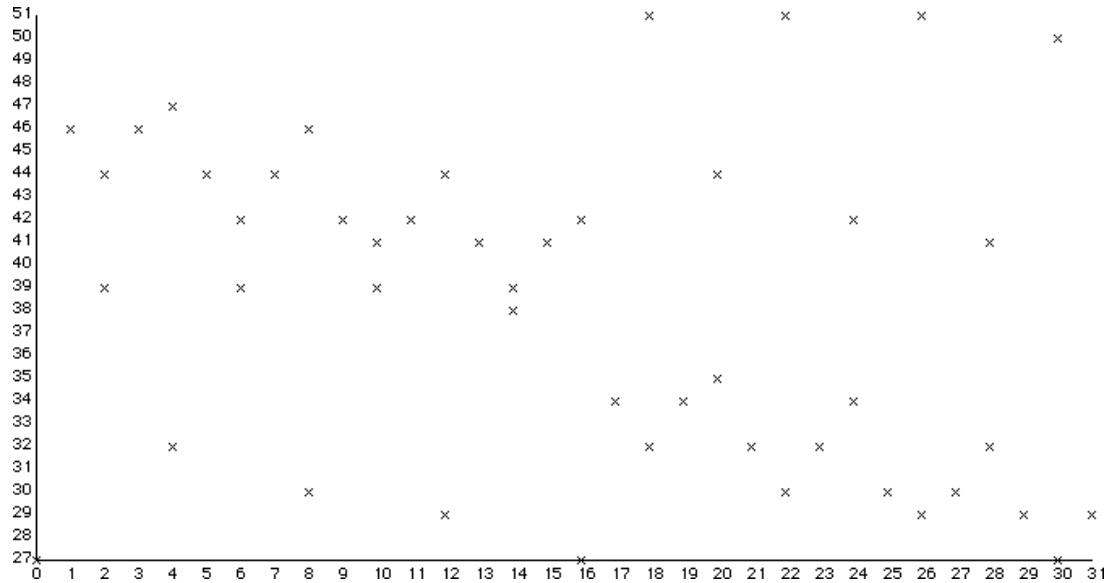
(b) It cannot be used for finding patterns in unvoiced polyphonic music.

5. Representing music using multidimensional datasets (1)



⟨onset time, chromatic pitch, morphetic pitch, duration, voice⟩

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



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. All these problems simply vanish when we abandon the string-based approach in favour of one where the music is represented as a multidimensional dataset.

4. A multidimensional dataset is simply a set of position vectors or datapoints in a Cartesian space with any number of dimensions. The algorithms that we're going to describe work with datasets of any dimensionality and any size. Also the co-ordinates may take real values (which, of course, in an implementation would be represented as floating point values).

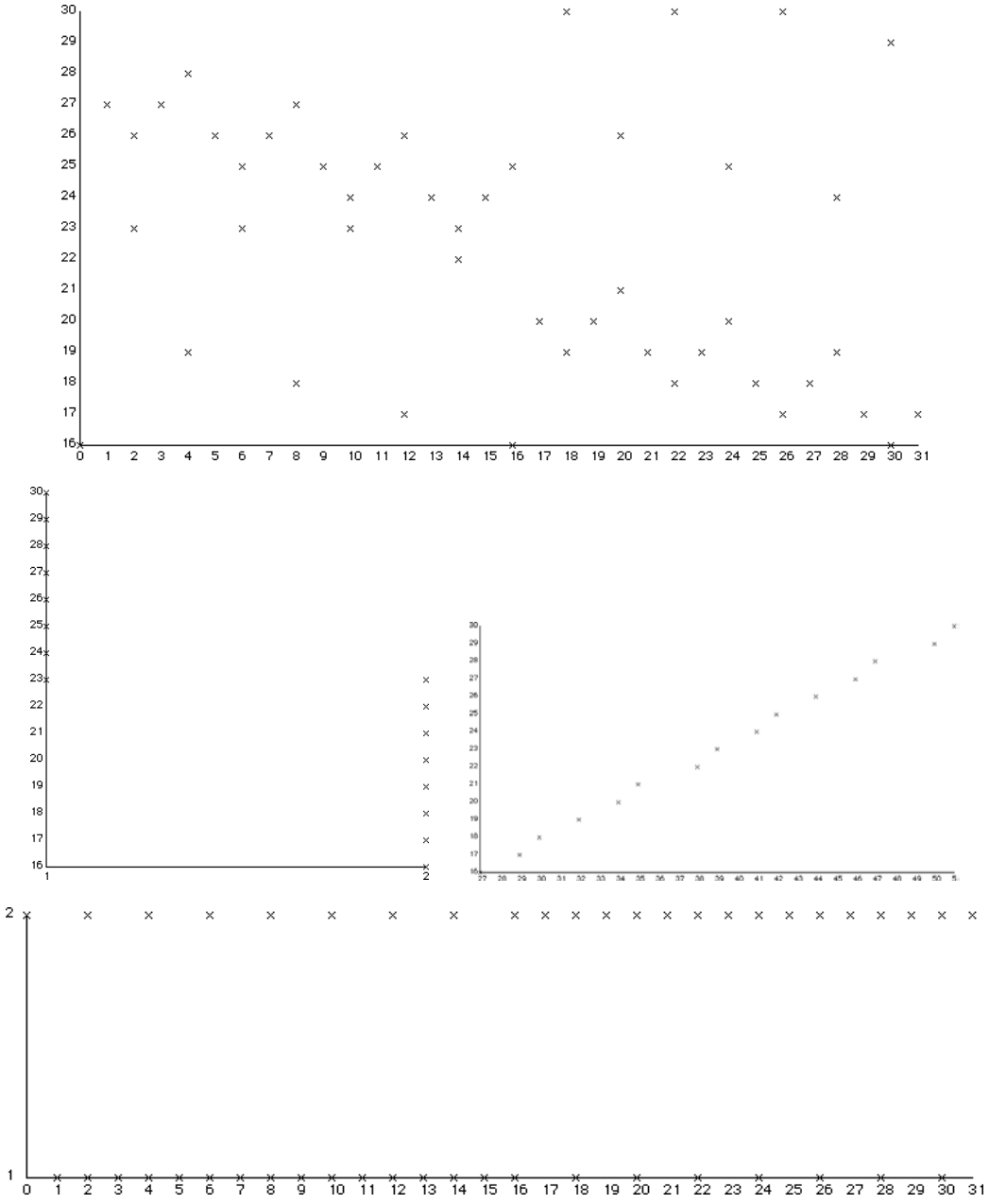
5. There are many possible appropriate ways of representing a piece of music as a multidimensional dataset and I'm going to use this example here to demonstrate some of the simpler possibilities.

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 co-ordinate 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.

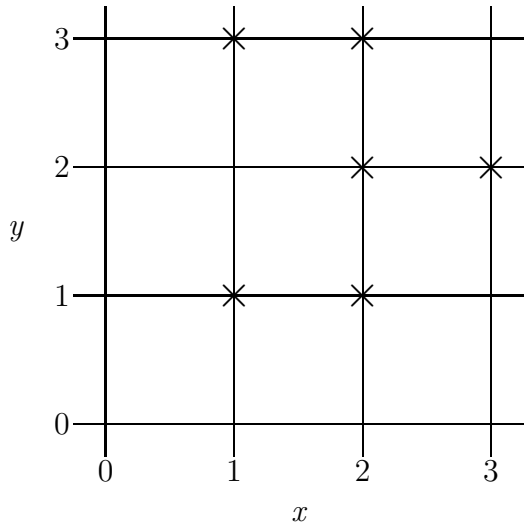
6. Representing music using multidimensional datasets (2)



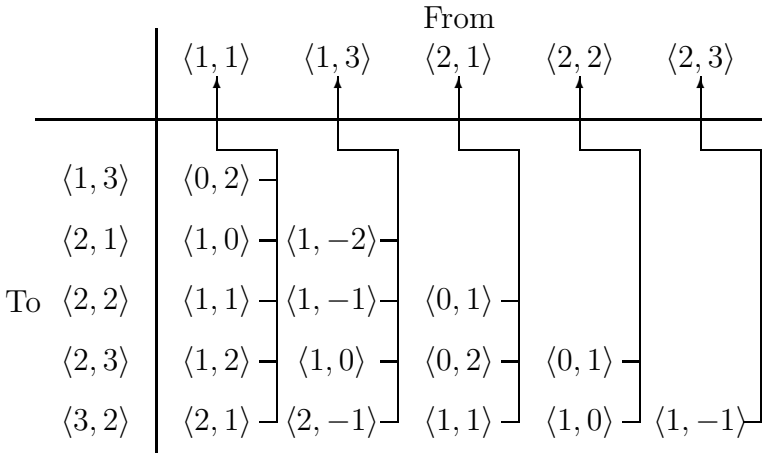
6. Representing music using multidimensional datasets (2)

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.

7. SIA: Discovering maximal repeated patterns in multidimensional datasets



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



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

1. `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.

2. 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 that consists of these three points $\{\langle 1, 1 \rangle, \langle 2, 2 \rangle, \langle 1, 3 \rangle\}$.

3. And the largest pattern that can be translated by the vector $\langle 1, 1 \rangle$ is the pattern that consists of these two points $\{\langle 1, 1 \rangle, \langle 2, 1 \rangle\}$.

4. We say that a pattern is *translatable* by a given vector if it can be translated by the vector to give another pattern that is a subset of the dataset.

5. The *maximal translatable pattern* for a given vector is then the largest pattern that can be translated by the vector to give another pattern that is in the dataset.

6. `SIA` discovers all the non-empty maximal translatable patterns for a given dataset and it does it like this:

7. First, the dataset is sorted.

8. 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.]

9. `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.

10. 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).

11. 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.

12. 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.

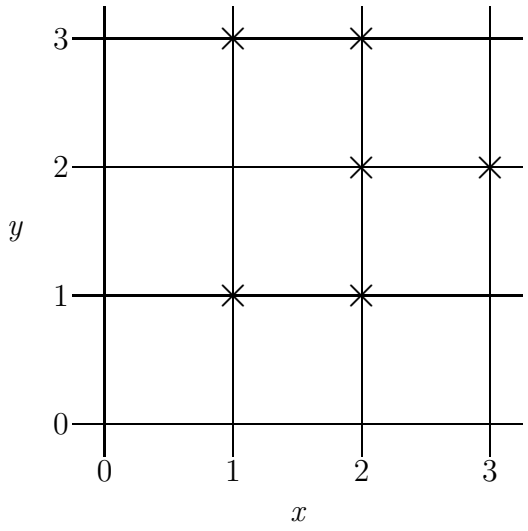
13. Note that each vector in this list is still linked to the datapoint at the head of its column in the vector table. Simply reading off all the datapoints attached to the adjacent occurrences of a given vector in this list gives us the maximal translatable pattern for that vector.

14. 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.

15. 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.

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

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



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

Time to find all occurrences of pattern $p = O(|p|n)$.

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

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

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

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

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

2. 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.]

3. 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.

4. So in `SIATEC` we actually compute this complete table here and we use the region below the leading diagonal to compute the maximal translatable patterns as in `SIA`.

5. 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.

6. 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.

7. Say we want to find all the occurrences of the pattern $\{\langle 1,1\rangle, \langle 2,1\rangle\}$ which is the maximal translatable pattern in this dataset for the vector $\langle 1,1\rangle$.

8. 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.

9. We know that the column of vectors under $\langle 1,1\rangle$ contains all the vectors that that datapoint can be translated by; and we know that the column under $\langle 2,1\rangle$ contains all the vectors that that datapoint can be translated by. So we know that the pattern $\{\langle 1,1\rangle, \langle 2,1\rangle\}$ is translatable only by those vectors that occur in both of these columns.

10. 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.

11. By exploiting the orderedness of this table, it's possible to find all the occurrences of a pattern p in a dataset of size n in a worst-case running time of $O(|p|n)$.

12. 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 then this implies

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

where $|p_i|$ is the cardinality of the ith pattern.

13. The overall worst-case running time of `SIATEC` is therefore

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

So the algorithm is $O(n^3)$ (or, in fact, $O(kn^3)$ for a k-dimensional dataset).

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

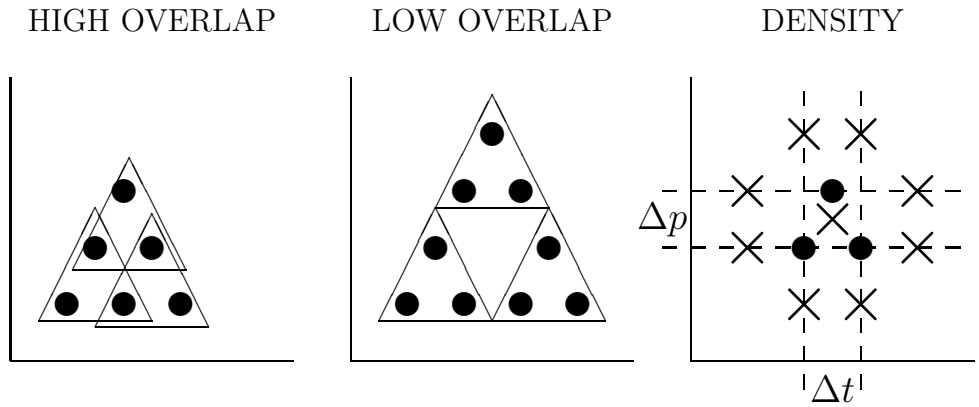
9. Isolating significant repetitions (1)

- 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.
  - Far fewer than 1000 of these are going to be analytically interesting.
  - Typically less than 1% of the patterns generated by `SIA` would be regarded as musically significant by an analyst or expert listener.
- Need systems that evaluate the output of `SIATEC` and isolate various classes of musically significant repetitions.

9. 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 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. So we can say that typically less than 1% of the patterns generated by `SIA` for a reasonably-sized piece of music would be regarded as musically interesting by an analyst or expert listener.
7. This means that we need to devise systems that evaluate the output of `SIATEC` and isolate various classes of musically interesting repetitions.

10. 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.
      i. Region defined as time interval spanned by pattern.
      ii. Region defined as time interval and pitch range spanned by pattern.

10. 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 is 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.

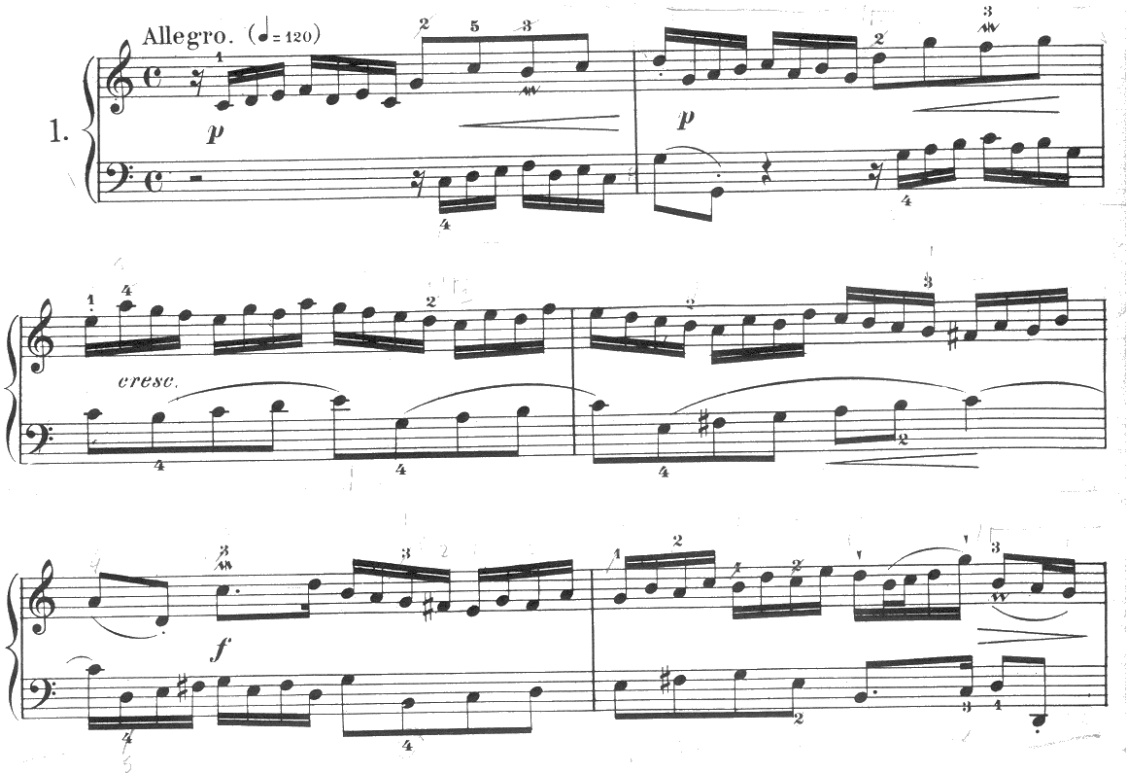
   (d) Cambouropoulos also uses these three quantities to select repeated patterns in his *GCTMS*.

   (e) *DENSITY* The denser or more compact the pattern, the better. There are a number of ways of measuring the density of a pattern:

      i. *Temporal density* Divide the number of notes in the pattern by the time interval spanned by the pattern.

      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 set of all notes that occur during the time interval spanned by the pattern. Or it could be all the notes that occur within the time interval *and* pitch range spanned by the pattern and so on.

11. Some preliminary results



11. Some preliminary results

1. So far, I've only tried out these heuristics on quite small datasets representing passages of music containing less than 200 notes. But the results have been quite encouraging.

2. 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 most "theme-like" using the heuristics I described on the previous slide is the seven-note subject of the Invention itself.

3. Here's what the passage sounds like with the occurrences of this pattern emphasized: [PLAY INVENTION-C.MID.]

4. [PUT ON SLIDE 2.]

5. 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".

6. [PUT ON SLIDE 3.]

7. 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".

8. I'll now hand you over to Geraint who will talk about his pattern-matching algorithm, `SIA(M)ESE`.

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.

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.