Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music

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

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

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

Department of Computer Science, University of Aarhus
Tuesday, 10 December 2002.

1. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music

1. The diversity of perceptually significant repetition in music.
2. Most repetitions in music are not interesting.
3. String-based 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. Heuristics for isolating perceptually significant repetitions in music.
9. `COSIATEC`: Using data-compression based on `SIATEC` for musical thematic analysis.
10. `SIAMESE`: Music information retrieval (pattern-matching) in multidimensional datasets.
11. Some possible directions for further work.

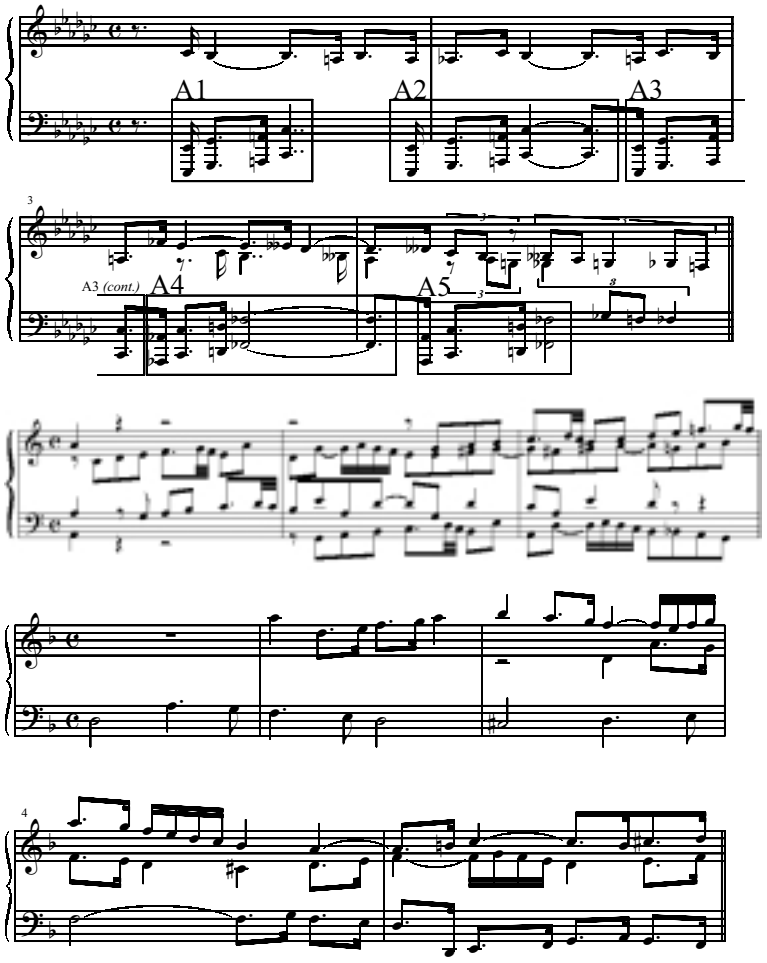
1. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music

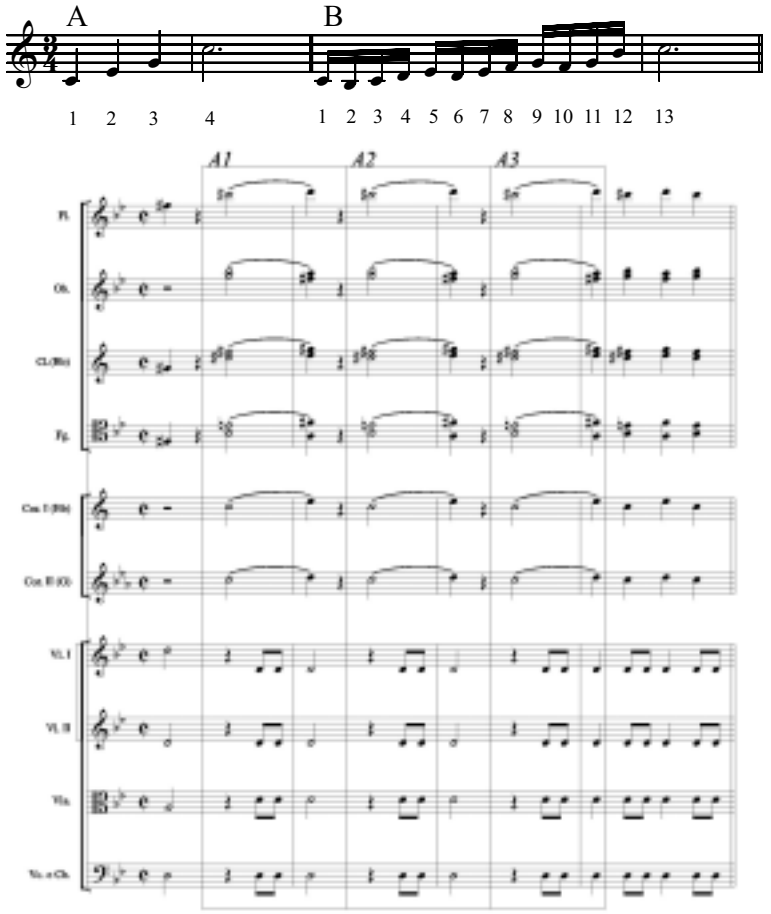
1. [THANK DEPARTMENT OF COMPUTER SCIENCE AND MARTIN ELMER JORGENSEN FOR INVITING ME TO GIVE A TALK.]
2. [E-MAIL LIST]
3. My name's David Meredith and I'm a Research Fellow at City University in London.
4. I'm going to talk to you about some work that I've been doing over the past couple of years in collaboration with Geraint Wiggins who is also at City University and Kjell Lemström at the University of Helsinki.
5. [PUT ON SLIDE 1].
6. In this project we've been focusing on developing algorithms for discovering perceptually significant repeated patterns in polyphonic music.
7. But the techniques we've developed can fairly straightforwardly be adapted for music information retrieval (pattern matching) and data-compression.
8. I'll begin by presenting some examples of perceptually significant repeated patterns in music which I hope will show you just how diverse this class of phenomena is.
9. Many music psychologists have noted that being able to identify perceptually significant repetitions is often extremely important for achieving a rich understanding of a piece. However, the vast majority of repetitions in a piece do, in fact, go unnoticed by listeners.
10. It seems that most previous approaches to repetition discovery in music have been based on the assumption that the music to be processed is represented in the form of a string or a set of strings.
11. I'll briefly review a couple of these string-based approaches and I'll show that there seem to be certain important types of musical repetition that are very difficult to find using these approaches.
12. It seems that if you want to find a wide range of different types of musical repeated pattern using a string-based approach, you generally have to run a variety of different algorithms on a number of different representations of the music.
13. In our work we've avoided these difficulties by adopting a *geometric* approach in which the music is represented as a multidimensional dataset—that is, a set of points in a Euclidean space.
14. We've found that by doing this we are 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; and

(c) essentially dispense with multiple representations because we can run the same small set of algorithms on various orthogonal projections of a single, rich multidimensional representation of the music.

15. I'll present two repetition discovery algorithms, `SIA` and `SIATEC`, that are based on this new 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.
16. I'll then briefly talk about what happens when you run these algorithms on music data.
17. 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.
18. So some post-processing is 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.
19. When these heuristics are incorporated into a data-compression algorithm based on `SIATEC` (which we call `COSIATEC`), we find that we can generate some quite interesting motivic and thematic music analyses.
20. I'll then briefly describe a pattern-matching algorithm based on `SIA`, which we call `SIAMESE`. This algorithm finds complete and partial matches of multidimensional query patterns in multidimensional datasets.
21. I think it's important to point out that although I'm going to be focusing on the musical applications of these 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.
22. I'll finish off 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 it.
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, or *parallelism*, as it is generally called by music psychologists and analysts.
3. However, the class of perceptually significant repetitions is a very diverse set. There are at least two reasons for this:
   (a) the patterns involved in such repetitions vary widely in their structural characteristics; and
   (b) 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, musical themes and motives can be truncated, augmented, diminished, inverted, reversed, embellished and so on.
4. I'll now show you a couple of examples that illustrate just how diverse the set of perceptually significant musical repetitions really is.
5. First, a 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 but 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 sounds like with the repeated bass-pattern emphasized. [PLAY BARBER-MODIFIED]. And here's what it sounds like without the bass-pattern emphasized. [PLAY BARBER WITHOUT MODIFIED BASS PART]
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. Repeated musical patterns also exhibit different types of *compactness*. For example, the repeated pattern in this Mozart example is what I call *temporally compact* because it contains all the notes that occur within the time period spanned by the pattern.

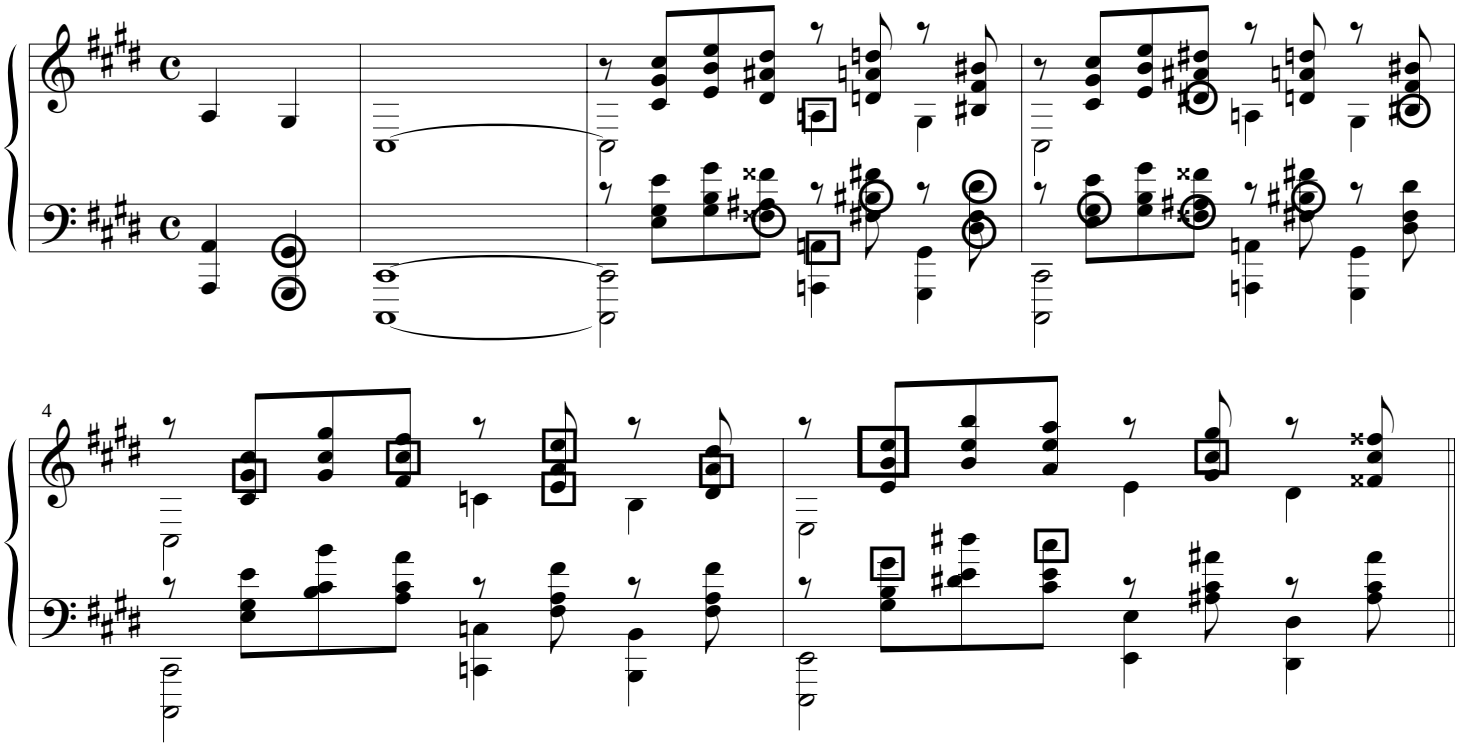
13. On the other hand, the rising bass pattern in this Barber example is *bounding box compact*—it contains all the notes in the piece that occur within the pattern's bounding box when the music is represented as a graph of chromatic pitch against time.

14. This Bach extract illustrates another type of compactness where each pattern contains all the notes that occur in a single voice within the time period spanned by the pattern.

15. In this example, taken from the beginning of *Contrapunctus VI* from Bach's *Die Kunst der Fuge*, we see themes being transformed by diminution and inversion. [ILLUSTRATE ON SLIDE] [PLAY CONTRAPUNCTUS]

16. And here's a simple example of a musical idea being embellished by replacing each long note by a sequence of shorter notes. [PLAY CANT-FIND-THIS-1]

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 like to demonstrate that although identifying the important repetitions in a piece significantly enriches a listener's understanding, not all repeated musical patterns 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 elliptical 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 BADPATTERN.MID].

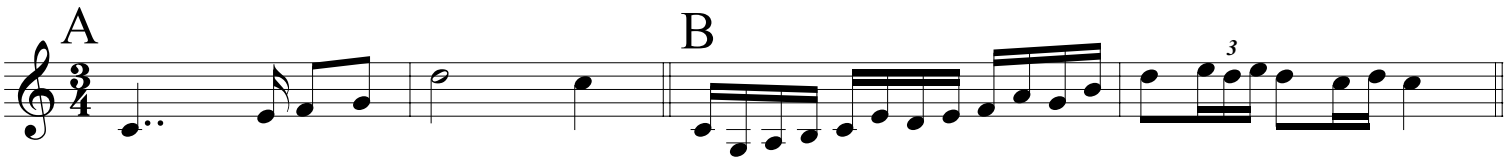
5. Clearly, this repetition is just an artefact that results from the other musically significant repetitions that are occurring in this passage such as, for example, the exact repetition of bar 3 in bar 4.

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 motivations behind our work has been to develop algorithms that extract only the interesting repeated patterns of a particular type from the music.

8. This involves formally characterising what it is about the interesting repetitions that distinguishes them from the many exact repetitions that the expert listener and analyst do not recognize as being important.

4. Previous approaches to repetition discovery in music



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 embellished repetitions.
Hsu *et al.* (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 and 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 can also be used to find repeated monophonic patterns in a polyphonic work in which each voice is represented as a string. Also, it is capable of finding repeated monophonic patterns which contain 'gaps'. However it suffers from a few weaknesses.

   (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 set so that it allows patterns of any size, the overall worst-case running time goes up to at least $O(n^4)$.

   (c) Third, like most string-based approaches to approximate pattern matching, it uses the edit-distance approach to compute the similarity between patterns. Unfortunately, such an approach is not typically capable of finding a match between a pattern and a highly-embellished variation on the pattern like the one shown here (CANT-FIND-THIS-2). This is because the edit distance between the two occurrences is rather large owing to the high number of insertions required to transform the plain version (A) into the embellished one (B).

   (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 *et al.* (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 only finds repeated *factors* and therefore cannot find patterns 'with gaps'. That is, it can only find the repetitions of a pattern if the pattern contains all the notes in the piece that occur during the time interval spanned by the pattern.

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:

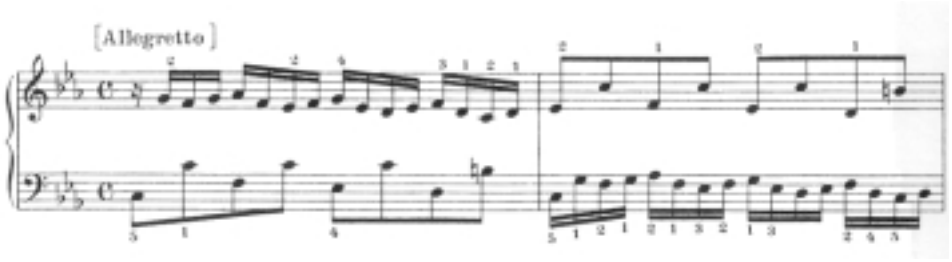
(a) It cannot find patterns with gaps.

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

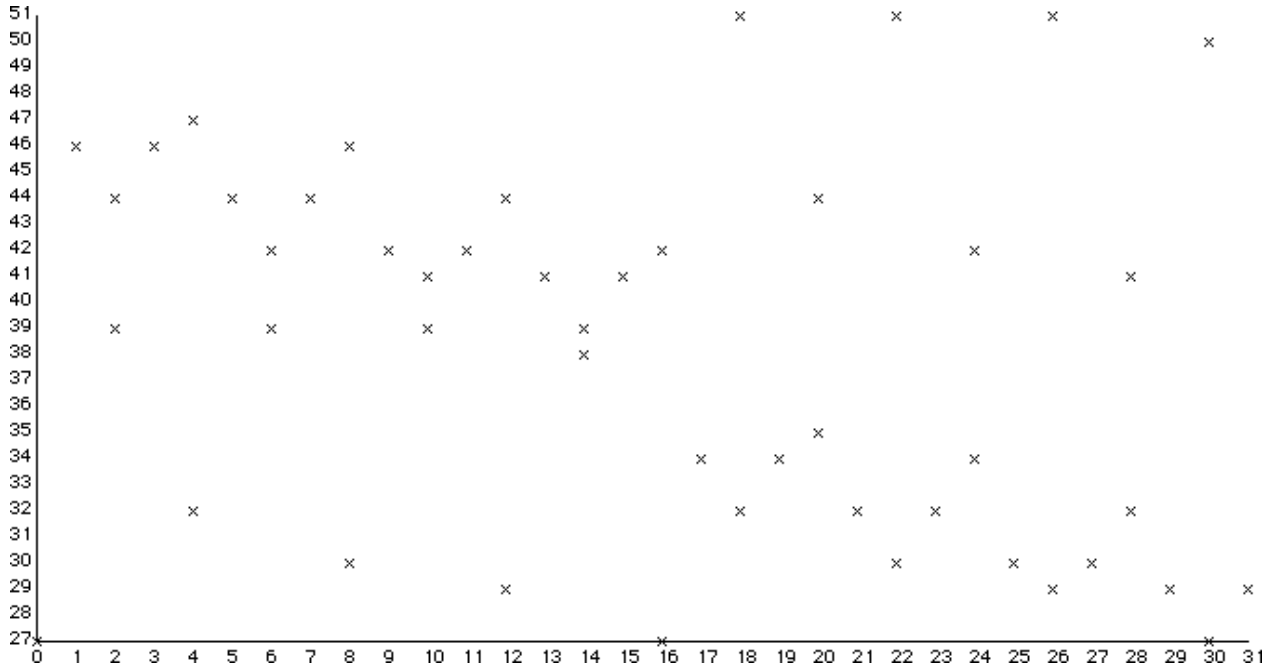
5. I'd also like to mention a recent approach described by Conklin and Anagnostopoulou (2001). In their method, a number of different string representations each representing a different 'viewpoint' on the music, are derived from a rich representation of the music to be analysed. They then discover repeated 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 (CANT-FIND-THIS-2) because the notes that are common to both occurrences of the pattern do not all fall on strong beats.

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)



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



5. Representing music using multidimensional datasets (1)

1. All the algorithms that I've just been talking about assume that the music is represented either as a 1-dimensional string of symbols or, in the case of polyphonic music, as a set of such symbol strings.

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 avoided these problems by adopting a *geometric* approach in which the music is represented as a multidimensional dataset.

4. A multidimensional dataset is just a finite set of position vectors or datapoints in a Euclidean space with a finite number of dimensions. Our algorithms work with datasets of any dimensionality and any size. Also the co-ordinates may take real values.

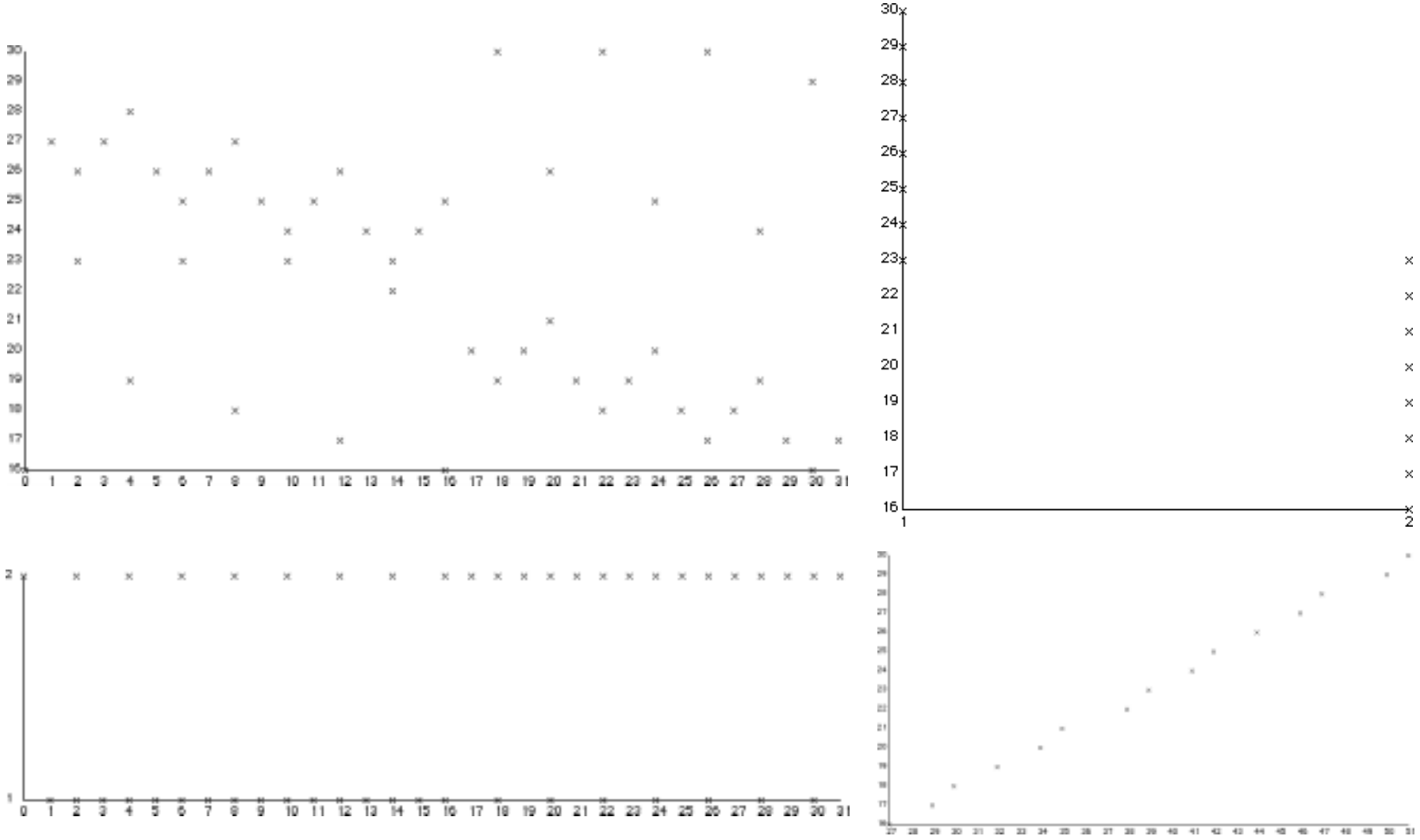
5. There are many possible appropriate ways of representing a piece of music as a multidimensional dataset and this is one fairly simple example.

6. At the top left here we have the first two bars of a Prelude from Bach's 48 Preludes and Fugues.

7. Then to the right of that, 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 in the score.

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.

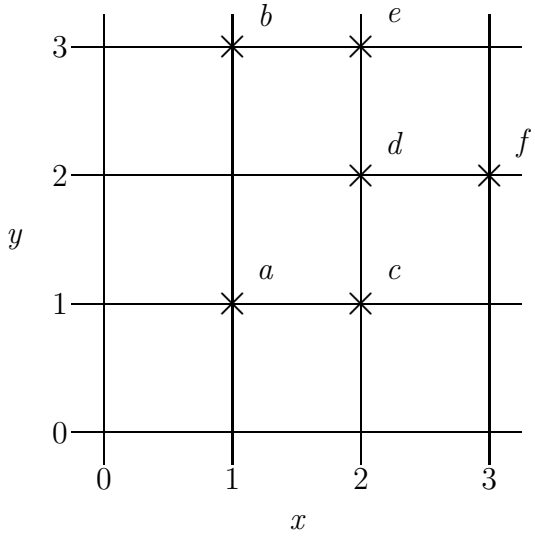
6. Representing music using multidimensional datasets (2)

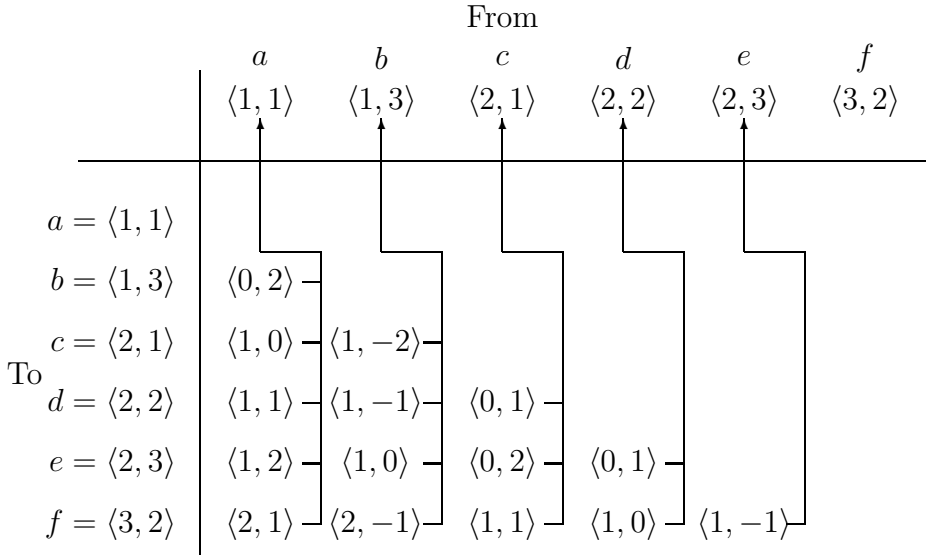


6. Representing music using multidimensional datasets (2)

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 (diatonic 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 represents 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





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

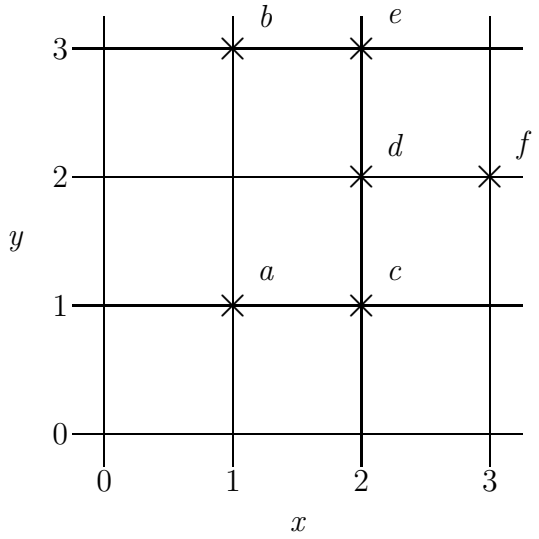
7. SIA: Discovering maximal repeated patterns in multidimensional datasets

1. 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. SIA then simply sorts the vectors in the table using a slightly modified version of merge sort that takes advantage of the fact that the columns in this table are already sorted.
13. This results in 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 $O(kn^2)$.

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



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

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 set 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

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

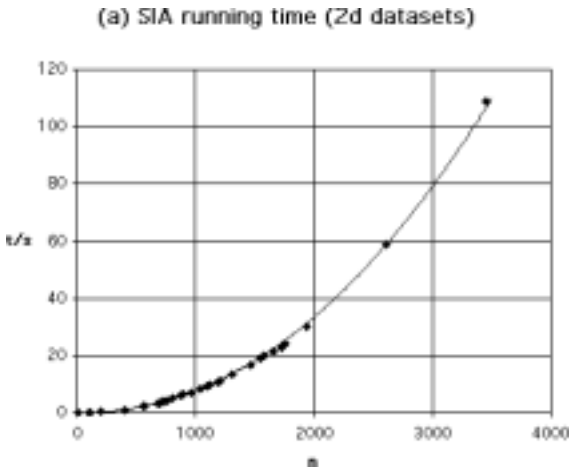
14. So the overall worst-case running time of `SIATEC` is

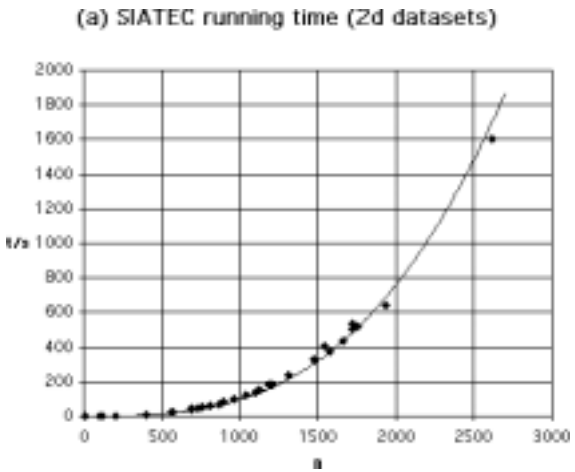
$$O\left(\sum_{i=1}^{l} km_i n\right) \leq O\left(\frac{kn^2(n-1)}{2}\right)$$

    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 graph on the left 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 graph on the right shows the running time of SIATEC on the same machine for the 2-dimensional datasets in the sample. In this graph, the smooth curve represents a running time of kn^3.
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.

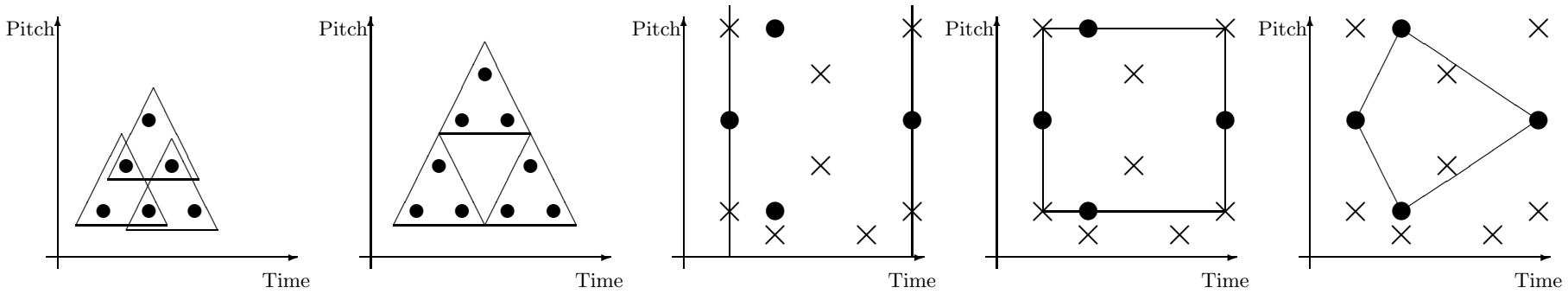
10. Isolating significant repetitions (1)

- 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; and
  - 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. Therefore, for all but the smallest datasets, `SIA` generates only a tiny fraction of all patterns in a 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)



Possible heuristics for finding "theme-like" patterns:

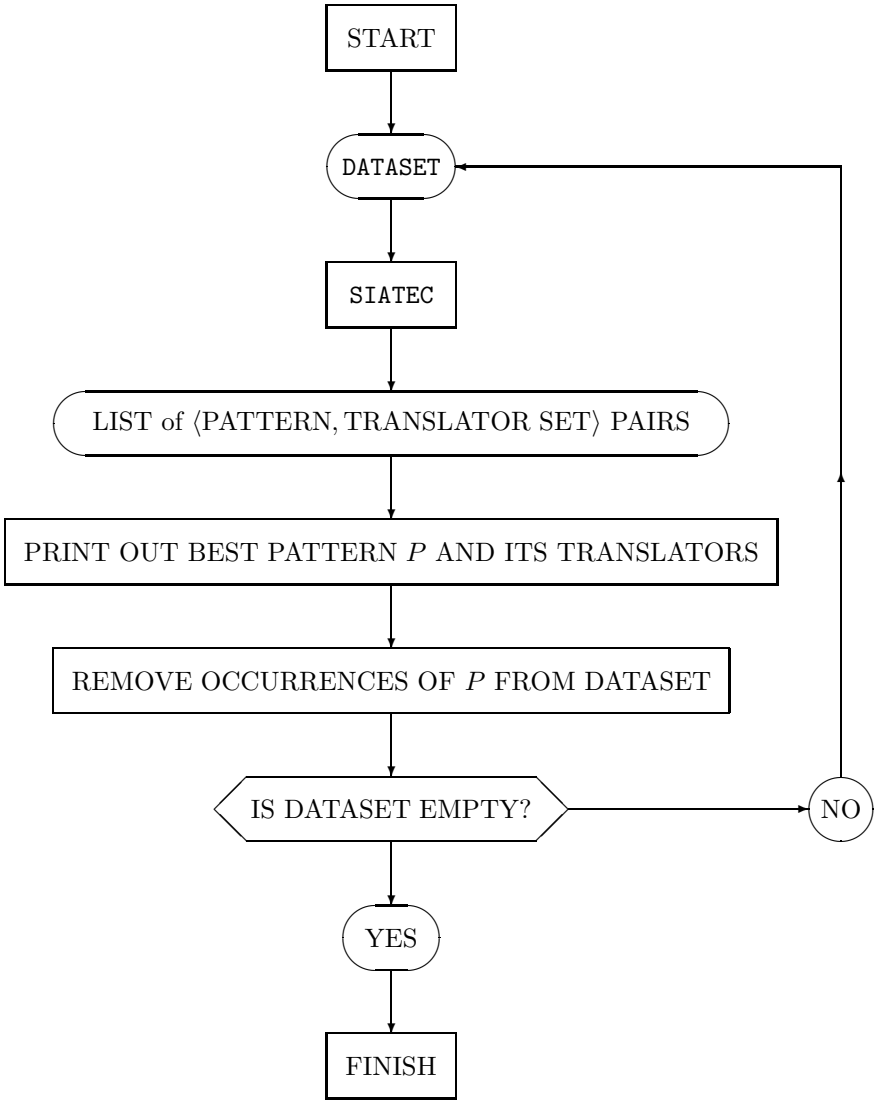
1. Coverage = Number of points covered by occurrences of the pattern.
2. Compactness = $\dfrac{\text{Number of notes in pattern}}{\text{Number of notes in piece in region spanned by pattern}}$.
3. Compression ratio = $\dfrac{\text{Coverage}}{\text{Size of Pattern}_{+}\text{Frequency of Occurrence}_{-1}}$.

11. Isolating significant repetitions (2)

1. Let's imagine that we want to carry out a thematic/motivic analysis of a piece. How can we isolate the theme-like and motive-like patterns?
2. Here are some suggestions for heuristics that seem to be useful for doing this.
3. First of all, it seems useful to define the concept of *coverage*. I define the *coverage* of a pattern to be the number of datapoints in the dataset that are members of occurrences of the pattern.
4. For example, for this triangular pattern in this dataset [LEFTMOST GRAPH] the coverage would be 6, but in this dataset, the coverage of the triangular pattern would be 9.
5. Note that coverage is generally greater for patterns whose occurrences overlap less. It is also generally greater for larger patterns and for those that occur more often.
6. In general, it seems that the most theme-like and motive-like patterns in music have relatively high coverage.
7. Next I define the concept of *compactness* to be the ratio of the number of points in the pattern to the total number of points in the dataset that occur within the region spanned by the pattern within a particular representation.
8. One can define 'the region spanned by a pattern' in a number of different ways:
9. For example, in this case here [THIRD GRAPH FROM LEFT] we've defined it to be the segment defined by the time-period spanned by the pattern. If we define it in this way, then this pattern here consisting of the four round points would have a compactness value of $\frac{4}{12} = \frac{1}{3}$.
10. Alternatively, one could define the region spanned by a pattern to be the bounding-box or the convex hull of the pattern in the pitch-against-onset-time graph of the piece. If we use the bounding box, as in this example [FOURTH GRAPH] then the compactness of this pattern here would be $\frac{4}{10} = \frac{2}{5}$. If we use the convex hull, as shown here [FIFTH GRAPH] then the compactness value would be $\frac{4}{6} = \frac{2}{3}$.
11. Typically, at least one occurrence of a theme-like pattern will have a high compactness value, even if the other occurrences are highly embellished.
12. Another interesting heuristic that seems to be useful for isolating theme-like patterns is the compression ratio that can be achieved by representing the set of points covered by all occurrences of the pattern by specifying simply one occurrence of the pattern and all the vectors by which the pattern can be translated.
13. For example, in this example here (left-hand figure), the set of points covered by the occurrences of the triangular pattern can be represented by specifying the points in one occurrence of the pattern and the two translation vectors that map that occurrence onto the other two occurrences of the pattern in this dataset.

14. That is, we can represent these 6 datapoints using 3 position vectors and 2 translation vectors, thus achieving a compression ratio of $\frac{6}{5}$. For the same pattern in this dataset [SECOND GRAPH], the compression ratio would be $\frac{9}{5}$.

12. COSIATEC

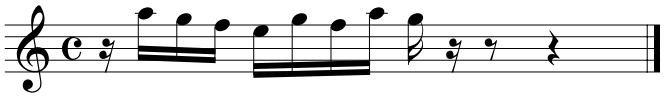

START
DATASET
SIATEC
LIST of ⟨PATTERN, TRANSLATOR SET⟩ PAIRS
PRINT OUT BEST PATTERN P AND ITS TRANSLATORS
REMOVE OCCURRENCES OF P FROM DATASET
IS DATASET EMPTY?
NO
YES
FINISH

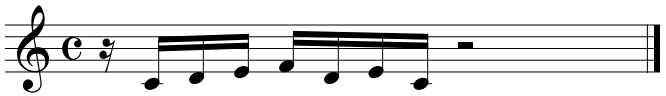
12. COSIATEC

1. COSIATEC is a compression algorithm based on SIATEC.
2. This flow-chart describes how it works.
3. First we run SIATEC on the dataset to be compressed. This generates a list of $\langle \text{PATTERN}, \text{TRANSLATOR SET} \rangle$ pairs.
4. The translator set for each pattern contains all the vectors by which the pattern is translatable within the dataset apart from the zero vector. In general, this gives a more efficient representation of the set of points covered by the occurrences of the pattern.
5. Then the heuristics that I described on the previous slide—compression ratio, coverage and compactness—are used to choose the 'best' pattern P and this pattern is printed out together with its translator set.
6. Then all the points covered by P—that is all the points that are members of occurrences of P—are removed from the dataset.
7. Then if the dataset is empty we finish, but if it isn't, we again run SIATEC on it and repeat the cycle.
8. The result is a print out of the 'best' pattern and its translators for each iteration of the cycle, and this printout is, in general, a compressed representation of the input dataset.
9. Obviously, the degree of compression achieved depends directly on the amount of repetition in the dataset.

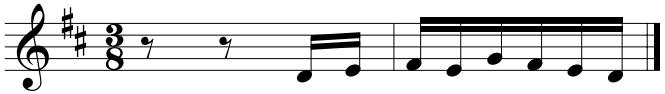
13. Running `COSIATEC` on music data

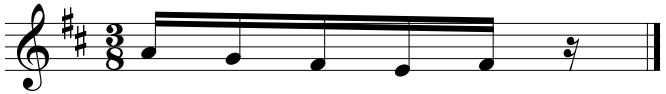
BWV772



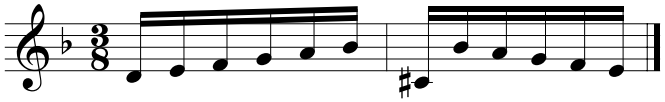


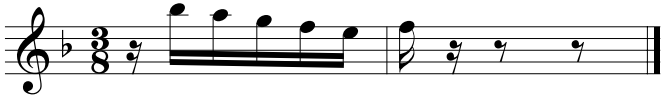
BWV774





BWV775

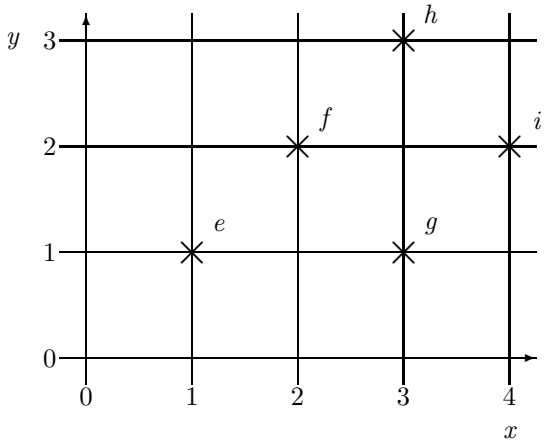


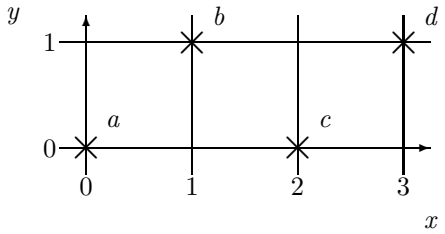


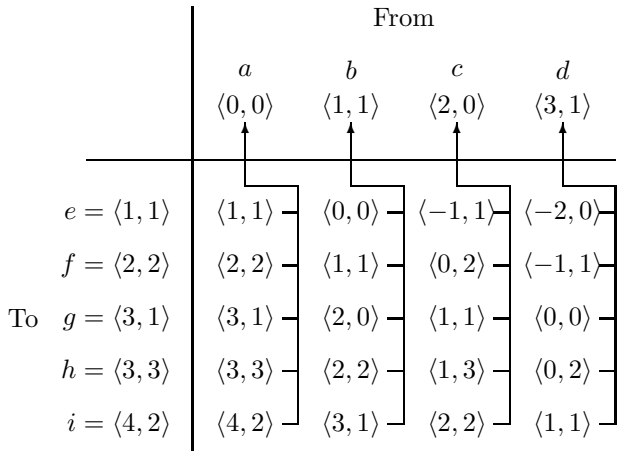
13. Running `COSIATEC` on music data

1. I've run COSIATEC on morphetic-pitch against onset-time representations of all 15 of Bach's Two-part Inventions and the results are quite encouraging.
2. In particular, it seems that the patterns that achieve the highest compression ratios on the early iterations of `COSIATEC` quite often correspond to the most important themes and motives in the music.
3. I've got a couple of examples here which I'll play to you. [PLAY EXAMPLES]

14. SIAMESE: Pattern matching in multidimensional datasets







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

14. SIAMESE: 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 SIAMESE. I'll now briefly describe how this algorithm works.
2. SIAMESE 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 SIAMESE [SHOW ON SLIDE].
4. In this case, SIAMESE 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.

15. Possible directions for further work

- Versions of `SIA` and `SIATEC`, `COSIATEC` and `SIAMESE` 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 further heuristics and algorithms for isolating various classes of perceptually significant repetition.
- Developing applications in specific domains (e.g., music, images, video, bioinformatics):
  - `SIA`, `SIATEC` and `COSIATEC`:
    - Data compression.
    - Database indexing.
    - Data mining.
  - `SIAMESE`:
    - Information retrieval.
    - Computer-based learning systems.

15. Possible directions for further work

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

- Algorithms are the subject of a patent submitted on 23 May 2001.
- Further information: http://www.titanmusic.com
- e-mail: dave@titanmusic.com

References

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

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

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

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.

Meredith, D., Lemström, K., and Wiggins, G. A. (2002). Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*. To appear.

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