COSIATEC AND SIATECCompress: PATTERN DISCOVERY BY GEOMETRIC COMPRESSION

David Meredith
Aalborg University
dave@titanmusic.com

ABSTRACT

Three versions of each of two greedy compression algorithms, COSIATEC and SIATECCompress, were run on the JKU Patterns Development Database. Each algorithm takes a point-set representation of a piece of music as input and computes a compressed encoding of the piece in the form of a union of translational equivalence classes of maximal translatable patterns. COSIATEC iteratively uses the SIATEC algorithm to strictly partition the input set into the covered sets of a set of MTP TECs. On each iteration, COSIATEC finds the "best" TEC and then removes its covered set from the input dataset. SIATECCompress runs SIATEC just once to get a list of MTP TECs and then selects a subset of the "best" TECs that is sufficient to cover the input dataset. Both algorithms select TECs primarily on the basis of compression ratio and compactness.

1. INTRODUCTION

In this paper, I present two greedy compression algorithms, COSIATEC and SIATECCompress, designed specifically to compute structural descriptions (i.e., analyses) of pieces of music. Both algorithms are based on the SIA and SIATEC algorithm described by Meredith, Lemström and Wiggins [8]. Each algorithm takes a point-set representation of a musical piece as input and computes a compact encoding of the piece in the form of a set of *translational equivalence classes* of *maximal translatable patterns*. COSIATEC generates a strict partitioning of the input dataset, whereas the sets of pattern occurrences computed by SIATECCompress may share points (i.e., notes).

Both algorithms are founded on the hypothesis that the best ways of understanding a piece of music are those that are represented by the shortest descriptions of the piece. In other words, they are designed to explore the notion that music analysis is effectively just music compression.

2. USING POINT SETS TO REPRESENT MUSIC

In the algorithms described below, it is assumed that the piece of music to be analysed is represented in the form of a multi-dimensional point set called a *dataset*, as described by Meredith *et al.* [8]. Although these algorithms work with datasets of any dimensionality, it will be assumed here that each dataset is a set of two-dimensional points, $\langle t, p \rangle$, where each point represents a single note or sequence of tied notes whose onset time is t in tatums and whose *morphetic pitch* [6–8] is p. If morphetic pitch information is not available (e.g., because the data is in MIDI format), then (at least for Western tonal music) it can be very reliably computed from chromatic pitch (i.e., MIDI note number) using an algorithm such as PS13s1 [6, 7].

3. MAXIMAL TRANSLATABLE PATTERNS

I shall use the term *pattern* to refer to any subset of a dataset. Suppose D is a dataset and $P_1, P_2 \subseteq D$. The two patterns, P_1, P_2, are said to be *translationally equivalent*, denoted by $P_1 \equiv_{\mathrm{T}} P_2$, if and only if there exists a vector v, such that P_1 translated by v is equal to P_2. That is,

$$P_1 \equiv_{\mathrm{T}} P_2 \iff (\exists v \mid P_2 = P_1 + v) . \tag{1}$$

Given a vector, v, then the *maximal translatable pattern* (MTP) for v in the dataset, D, is defined and denoted as follows:

$$\mathrm{MTP}(v, D) = \{p \mid p \in D \wedge p + v \in D\} \tag{2}$$

where $p + v$ is the point that results when one translates p by the vector v. In other words, the MTP for a vector v in a dataset D is the set of points in D that can be translated by v to give other points that are also in D.

The notion that COSIATEC and SIATECCompress can be used to discover the patterns in a piece of music that an analyst or a listener finds important, is founded upon the hypothesis that these patterns correspond in some way to MTPs in the pitch-time dataset representation of the piece. Meredith *et al.* [8] describe an algorithm called SIA for discovering all the MTPs in a dataset.

4. TRANSLATIONAL EQUIVALENCE CLASSES

When analysing a piece of music, we typically want to find *all the occurrences* of an interesting pattern, not just one occurrence. Given a pattern, P, in a dataset, D, the *translational equivalence class* (TEC) of P in D is defined and denoted as follows:

$$\mathrm{TEC}(P, D) = \{Q \mid Q \equiv_{\mathrm{T}} P \wedge Q \subseteq D\} . \tag{3}$$

We can also define the *covered set* of a TEC, T, denoted by $\mathrm{COV}(T)$, to be the union of the occurrences in the TEC. That is,

$$\mathrm{COV}(T) = \bigcup_{P \in T} P \,. \qquad (4)$$

Here we will be particularly concerned with *MTP TECs*—that is, the translational equivalence classes of the maximal translatable patterns in a dataset. Meredith *et al.* [8] describe an algorithm called SIATEC that uses SIA to find all the MTPs and then goes on to find the TEC of each of these MTPs (i.e., it finds all the (exact) occurrences of all the MTPs).

A TEC is a set of patterns that are all translationally equivalent to each other. Suppose a TEC, T, contains n occurrences of a pattern containing m points. There are at least two ways in which one can specify T. First, one can list each of the n occurrences in T explicitly by listing all of the m points in each occurrence. This requires one to write down mn 2-dimensional points or $2mn$ integers. Alternatively, one can explicitly list the m points in just one of the n occurrences, P, and then give the $n-1$ vectors required to map P onto the other occurrences. This requires one to write down m 2-dimensional points and $n-1$, 2-dimensional vectors—that is, $2(m+n-1)$ integers. If n and m are both greater than one, then $2(m+n-1)$ is less than $2mn$, implying that the second method of specifying a TEC gives us a *compressed* encoding of the TEC (and therefore also of its covered set). Thus, in principle, if a dataset contains repeated (i.e., translationally equivalent) patterns, it may be possible to encode the dataset in a compact manner by representing it as the union of the covered sets of a set of TECs, where each TEC, T, is encoded as an ordered pair, $\langle P, V\rangle$, where P is one occurrence in T and V is the set of vectors that map P onto the other occurrences in T. When a TEC, $T = \langle P, V\rangle$, is represented in this way, we call P the *pattern* and V the *translator set* of the TEC.

5. THE COSIATEC ALGORITHM

COSIATEC [5, 9] (see Figure 1) is a greedy compression algorithm, based on SIATEC, that takes a dataset, D, as input and computes a compressed encoding of D in the form of an ordered set of MTP TECs, $\mathbf{T}$, such that

$$D = \bigcup_{T \in \mathbf{T}} \mathrm{COV}(T) \qquad (5)$$

and, for all $T_1, T_2 \in \mathbf{T}$, $T_1 \neq T_2$,

$$\mathrm{COV}(T_1) \cap \mathrm{COV}(T_2) = \emptyset \,. \qquad (6)$$

In other words, COSIATEC partitions a dataset D into the covered sets of a set of MTP TECs. If each of these MTP TECs is represented as a $\langle$pattern, translator set$\rangle$ pair, then this description of the dataset as a set of TECs is typically shorter than an *in extenso* description in which the points in the dataset are simply listed explicitly.

COSIATEC begins by making a copy of the input dataset which it stores in the variable P (line 1). Then, on each iteration of the **while** loop (lines 4–7), the algorithm finds the "best" MTP TEC in P, T^*, appends this TEC to $\mathbf{T}$ and then removes the set of points covered by T^* from P. When P is empty, the algorithm terminates, returning the list of MTP TECs, $\mathbf{T}$. The sum of the number of translators and the number of points in this output encoding is never more than the number of points in the input dataset and can be much less than this if there are many repeated patterns in the input dataset.

```
COSIATEC($D$)
1  $P \leftarrow$ COPY($D$)
2  $T^* \leftarrow$ nil
3  $\mathbf{T} \leftarrow \langle\rangle$
4  while $P \neq \emptyset$
5      $T^* \leftarrow$ GETBESTTEC($P, D$)
6      $\mathbf{T} \leftarrow \mathbf{T} \oplus \langle T^* \rangle$
7      $P \leftarrow P \setminus \mathrm{COV}(T^*)$
8  return $\mathbf{T}$
```

Figure 1. The COSIATEC algorithm.

```
GETBESTTEC($P, D$)
1   $\mathbf{V} \leftarrow$ COMPUTEVECTORTABLE($P$)
2   $\mathbf{MCPs} \leftarrow$ COMPUTEMTPCISPAIRS($\mathbf{V}$)
3   $mcp \leftarrow$ nil
4   $T^* \leftarrow$ nil
5   for $i \leftarrow 0$ to $|\mathbf{MCPs}| - 1$
6       $mcp \leftarrow \mathbf{MCPs}[i]$
7       $T \leftarrow$ GETTECFORMTP($mcp, \mathbf{V}, P$)
8       $conj \leftarrow$ GETCONJ($T$)
9       $T \leftarrow$ REMREDTRAN($T$)
10      $conj \leftarrow$ REMREDTRAN($conj$)
11      if $T^* =$ nil $\vee$ ISBETTERTEC($T, T^*$)
12          $T^* \leftarrow T$
13      if ISBETTERTEC($conj, T^*$)
14          $T^* \leftarrow conj$
15  return $T^*$
```

Figure 2. The GETBESTTEC algorithm.

Given an input dataset, D, and what remains of a copy, P, of this dataset after the removal of zero or more MTP TEC covered sets, the COSIATEC algorithm finds the "best" MTP TEC in P (line 5), using the GETBESTTEC algorithm shown in Figure 2. In lines 1–2 of GETBEST-TEC, the SIA algorithm is used to find all the MTPs in the dataset. The first step in this process is to compute a so-called *vector table*, $\mathbf{V}$, which is a two-dimensional array of ordered triples,

$$\mathbf{V}[i][j] = \langle p_i - p_j, p_j, j \rangle \,,$$

where $p_i - p_j$ is the vector from point p_j to p_i and $p_k = \mathbf{P}[k]$, where $\mathbf{P}$ is an ordered set that only contains every element in P, sorted into lexicographical order.

Having computed the vector table, $\mathbf{V}$, the MTPs are found by sorting the triples in $\mathbf{V}$, lexicographically by their vectors (i.e., their first elements), and then scanning this sorted list once: each MTP is then equal to the points associated with a run of consecutive triples with the same vector in this sorted list. This is accomplished in line 2 of GETBESTTEC using the COMPUTEMTPCISPAIRS algorithm, which is shown in Figure 3.

```
ComputeMtpCisPairs($\mathbf{V}$)
1   $\mathbf{W} \leftarrow$ SortByVector($\mathbf{V}$)
2   $\mathbf{MTPs} \leftarrow \langle\rangle$
3   $\mathbf{CISs} \leftarrow \langle\rangle$
4   $v \leftarrow \mathbf{W}[0][0]$
5   $\mathbf{mtp} \leftarrow \langle \mathbf{W}[0][1] \rangle$
6   $\mathbf{cis} \leftarrow \langle \mathbf{W}[0][2] \rangle$
7   for $i \leftarrow 1$ to $|\mathbf{W}| - 1$
8       $\mathsf{vpi} \leftarrow \mathbf{W}[i]$
9       if $\mathsf{vpi}[0] = v$
10          $\mathbf{mtp} \leftarrow \mathbf{mtp} \oplus \langle \mathsf{vpi}[1] \rangle$
11          $\mathbf{cis} \leftarrow \mathbf{cis} \oplus \langle \mathsf{vpi}[2] \rangle$
12      else
13          $\mathbf{MTPs} \leftarrow \mathbf{MTPs} \oplus \langle \mathbf{mtp} \rangle$
14          $\mathbf{CISs} \leftarrow \mathbf{CISs} \oplus \langle \mathbf{cis} \rangle$
15          $\mathbf{mtp} \leftarrow \langle \mathsf{vpi}[1] \rangle$
16          $\mathbf{cis} \leftarrow \langle \mathsf{vpi}[2] \rangle$
17          $v \leftarrow \mathsf{vpi}[0]$
18  $\mathbf{MTPs} \leftarrow \mathbf{MTPs} \oplus \langle \mathbf{mtp} \rangle$
19  $\mathbf{CISs} \leftarrow \mathbf{CISs} \oplus \langle \mathbf{cis} \rangle$
20  $\mathbf{MCPs} \leftarrow \langle\rangle$
21  for $i \leftarrow 0$ to $|\mathbf{MTPs}| - 1$
22      $\mathbf{MCPs} \leftarrow \mathbf{MCPs} \oplus \langle \langle \mathbf{MTPs}[i], \mathbf{CISs}[i] \rangle \rangle$
23  return $\mathbf{MCPs}$
```

Figure 3. The ComputeMtpCisPairs algorithm.

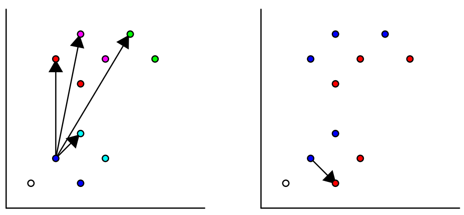

Figure 4. A pair of conjugate TECs. Note that the pattern of blue points in the right-hand figure consists of the upper left point of each pattern in the TEC in the left-hand figure.

The ComputeMtpCisPairs algorithm (Figure 3) first sorts the triples in the vector table, $\mathbf{V}$, into increasing lexicographical order by their vectors. The resulting ordered set of triples is stored in the variable $\mathbf{W}$ (see line 1). In lines 2–19 of this algorithm, two lists are constructed, $\mathbf{MTPs}$ and $\mathbf{CISs}$. $\mathbf{MTPs}$ contains all the MTPs in the dataset, each MTP being represented as an ordered set of points in lexicographical order. $\mathbf{CISs}$ contains, for each MTP, a list of the indices of the columns in the vector table corresponding to the points in the MTP. In lines 20–22 of ComputeMtpCisPairs, a list of $\langle \mathbf{mtp}, \mathbf{cis} \rangle$ pairs is constructed by combining corresponding elements in $\mathbf{MTPs}$ and $\mathbf{CISs}$.

In lines 5–14 of GetBestTEC, the **for** loop iterates over this ordered set of $\langle \mathbf{mtp}, \mathbf{cis} \rangle$ pairs computed by ComputeMtpCisPairs. For each pair, the TEC of the MTP is computed in line 7 using the technique employed in the SIATEC algorithm [8]. Then, in line 8, the *conjugate* TEC [1] is computed for each MTP TEC found in line 7. The concept of a conjugate TEC is illustrated in Figure 4. Given a TEC, $T = \langle P, V \rangle$, the conjugate of T is denoted and defined as follows:

$$\text{GetConj}(T) = \langle P', V' \rangle \tag{7}$$

where, if p_0 is the lexicographically first point in P,

$$P' = \{p_0\} \cup \{p_0 + v \mid v \in V\}\ , \tag{8}$$

and

$$V' = \{p - p_0 \mid p \in P\} \setminus \{\langle 0, 0 \rangle\}\ . \tag{9}$$

Given a pair of conjugate TECs, one may be "better" than the other (e.g., because its pattern might be more compact).

In lines 9 and 10 of GetBestTEC, redundant translators are removed from both the TEC, T, and its conjugate using the RemRedTran algorithm. A translator is defined to be *redundant* if it can be removed from the translator set of a TEC without changing the covered set of the TEC. Ideally, in order to get the most compact description of the covered set of a TEC, one would want to remove as many redundant translators as possible. However, in general, finding the smallest subset of the translator set of a TEC that is sufficient to generate the TEC's covered set is an NP-hard problem. In the implementation of COSIATEC submitted to the MIREX 2013 competition, a greedy approximation algorithm is used to remove as many redundant translators as possible from a TEC within a reasonable running time.

Finally, in lines 11–14 of GetBestTEC, each MTP TEC and its conjugate are compared with the "best" TEC so far and replace it if they are deemed superior to it by the IsBetterTEC function, defined in Figure 5. This function takes two TECs as its arguments and returns true if the first is "better than" the second. In lines 1–2 of IsBetterTEC, the compression ratio of the two TECs are compared. If $\mathrm{P}(T)$ and $\mathrm{V}(T)$ are defined to return the pattern and translator set of a TEC, T, respectively, then the compression ratio of a TEC is defined as follows:

$$\text{CompRatio}(T) = \frac{|\mathrm{COV}(T)|}{|\mathrm{P}(T)| + |\mathrm{V}(T)| - 1}\ . \tag{10}$$

If the two TECs to be compared have the same compression ratio, then they are compared for bounding-box *compactness* (lines 3–4 of IsBetterTEC) [8]. The bounding-box compactness of a TEC is the number of points in the TEC's pattern divided by the number of dataset points in the bounding box of this pattern. If the two TECs have the same compression ratio and compactness, the TEC with largest covered set is considered superior (lines 5–6). If the two covered sets are also the same size, then the TEC with the larger pattern is considered superior (lines 7–8). If the patterns are also the same size, then the TEC with the pattern that has the shorter temporal duration is considered superior (lines 9–10). Finally, if the two TECs also have the same temporal duration, then the TEC with the pattern whose bounding box has the smaller area is considered superior (lines 11–12).

6. THE SIATECCompress ALGORITHM

COSIATEC runs SIATEC on each iteration of its **while** loop. Since SIATEC has worst case running time $O(n^3)$

```
IsBetterTEC($T_1$, $T_2$)
1   if CompRatio($T_1$) $>$ CompRatio($T_2$)
2       return true
3   if Compactness($T_1$) $>$ Compactness($T_2$)
4       return true
5   if $|\text{COV}(T_1)| > |\text{COV}(T_2)|$
6       return true
7   if PatternSize($T_1$) $>$ PatternSize($T_2$)
8       return true
9   if PatternWidth($T_1$) $<$ PatternWidth($T_2$)
10      return true
11  if PatternArea($T_1$) $<$ PatternArea($T_2$)
12      return true
13  return false
```

Figure 5. The IsBetterTEC function.

```
SIATECCompress($D$)
1   $\mathbf{V} \leftarrow$ ComputeVectorTable($D$)
2   $\mathbf{MCPs} \leftarrow$ ComputeMtpCisPairs($\mathbf{V}$)
3   $\mathbf{MCPs} \leftarrow$ RemoveTranEquivMTPs($\mathbf{MCPs}$)
4   $\mathbf{T} \leftarrow$ ComputeTECs($D, \mathbf{V}, \mathbf{MCPs}$)
5   $\mathbf{T} \leftarrow$ AddConjugateTECs($\mathbf{T}$)
6   $\mathbf{T} \leftarrow$ RemRedTran($\mathbf{T}$)
7   $\mathbf{T} \leftarrow$ SortTECsByQuality($\mathbf{T}$)
8   return ComputeEncoding($D, \mathbf{T}$)
```

Figure 6. The SIATECCompress algorithm.

where n is the number of points in the input dataset, running COSIATEC on large datasets can be time-consuming (see Table 1 for some example running times). On the other hand, because COSIATEC strictly partitions the dataset into non-overlapping MTP TEC covered sets, it tends to achieve high compression ratios for many point-set representations of musical pieces (typically between 2 and 4 for a piece of classical or baroque music).

Like COSIATEC, the SIATECCompress algorithm shown in Figure 6 is a greedy compression algorithm based on SIATEC that computes an encoding of a dataset in the form of a union of TECs. SIATECCompress closely resembles the algorithm described by Forth [3,4], but is simpler and non-parametric. Like Forth's algorithm, but unlike COSIATEC, SIATECCompress runs SIATEC only *once* to get a list of TECs in decreasing order of quality (as defined by the IsBetterTEC function in Figure 5). It then works its way down this list, selecting TECs to include in the encoding, until the input dataset is covered. SIATECCompress does not generally produce as compact an encoding as COSIATEC, since the TECs in its output may share points. However, it is faster than COSIATEC and can therefore be used practically on much larger datasets.

The first steps in SIATECCompress are to compute a vector table and compute MTPs using the SIA algorithm, implemented in ComputeVectorTable and ComputeMtpCisPairs, as in the first two lines of GetBestTEC (see Figure 2). The next step (line 3 in Figure 6) is to remove MTPs from the list, $\mathbf{MCPs}$, that are translationally equivalent to MTPs that occur earlier in this list. This eliminates the possibility of the same TEC

```
ComputeEncoding($D, \mathbf{T}$)
1   $P \leftarrow \emptyset$
2   $\mathbf{E} \leftarrow \langle\rangle$
3   for $i \leftarrow 0$ to $|\mathbf{T}| - 1$
4       $T \leftarrow \mathbf{T}[i]$
5       $S \leftarrow \text{COV}(T)$
6       if $|S \setminus P| > |\text{P}(T)| + |\text{V}(T)| - 1$
7           $\mathbf{E} \leftarrow \mathbf{E} \oplus \langle T \rangle$
8           $P \leftarrow P \cup S$
9           if $|P| = |D|$
10              break
11  $R \leftarrow D \setminus P$
12  if $|R| > 0$
13      $\mathbf{E} \leftarrow \mathbf{E} \oplus \langle \text{AsTEC}(R) \rangle$
14  return $\mathbf{E}$
```

Figure 7. The ComputeEncoding algorithm.

being computed more than once in line 4. In line 5, the conjugate of each TEC found in line 4 is also added to the list of candidate TECs, $\mathbf{T}$. In line 6, redundant translators are removed from the translator set of each TEC in $\mathbf{T}$ and, in line 7, the resulting list of candidate TECs is sorted into decreasing order of quality using the IsBetterTEC comparator function. This ordered set of TECs is then given to the ComputeEncoding function (Figure 7), which computes a compact encoding of the input dataset.

7. RESULTS

Three versions of each of the two algorithms described above were run on the JKU Patterns Development Database [1] (JKU PDD) [2]. The results are shown in Table 1. The values in the table were computed using Tom Collins' MATLAB implementation of the metrics defined in [2], bundled with the JKU PDD.

Each row in Table 1 gives the results of running one version of an algorithm on one of the five pieces in the JKU PDD. The first column gives the name of the algorithm. Each name either has no suffix (e.g., "COSIATEC", "SIATECCompress") or one of the two suffixes, "BB" or "Segment". A name with no suffix indicates that the row shows the results of running the plain algorithm as described above, with the discovered patterns equal to the MTP TECs in the output encoding. A name with the suffix "BB" indicates that each occurrence within a TEC in the output of the algorithm is replaced with the set of dataset points in the bounding-box of the occurrence. A name with the suffix "Segment" indicates that each occurrence within a TEC in the output of the algorithm is replaced with the set of dataset points in the time segment spanned by the occurrence.

The following preliminary observations can be made from studying these results:

1. The algorithms generally score better on establishment recall than establishment precision; whereas each algorithm's occurrence recall and occurrence

[1] `https://dl.dropbox.com/u/11997856/JKU/JKUPDD-noAudio-Aug2013.zip`

Algorithm	Piece	Mono/Poly	Notes	n_P	n_Q	P_est	R_est	F1_est	P_occ(c=.75)	R_occ(c=.75)	F_1occ(c=.75)	P_3	R_3	TLF_1	runtime/ms	FFTP_est	FFP	P_occ(c=.5)	R_occ(c=.5)	F_1occ(c=.5)	P	R	F_1
COSIATEC	bach_wtc2f20	Mono	731	3.00	10.00	0.33	0.93	0.49	0.81	0.80	0.81	0.27	0.87	0.41	132389	0.67	0.37	0.81	0.80	0.81			
COSIATECBB	bach_wtc2f20	Mono	731	3.00	10.00	0.33	0.93	0.49	0.81	0.80	0.80	0.27	0.86	0.41	130718	0.67	0.37	0.81	0.80	0.80			
COSIATECSegment	bach_wtc2f20	Mono	731	3.00	10.00	0.33	0.93	0.49	0.81	0.80	0.80	0.27	0.86	0.41	130828	0.67	0.37	0.81	0.80	0.80			
SIATECCompress	bach_wtc2f20	Mono	731	3.00	20.00	0.25	0.62	0.36	1.00	0.86	0.93	0.20	0.53	0.29	37560	0.33	0.37	0.56	0.63	0.59			
SIATECCompressBB	bach_wtc2f20	Mono	731	3.00	20.00	0.32	0.64	0.43	0.86	0.60	0.71	0.24	0.54	0.33	38132	0.33	0.37	0.62	0.69	0.65			
SIATECCompressSegment	bach_wtc2f20	Mono	731	3.00	20.00	0.34	0.66	0.44	0.74	0.79	0.77	0.26	0.56	0.35	38939	0.33	0.37	0.61	0.80	0.69			
COSIATEC	bach_wtc2f20	Poly	733	3.00	9.00	0.33	0.77	0.46	0.73	0.77	0.75	0.32	0.73	0.44	80315	0.50	0.41	0.61	0.66	0.63			
COSIATECBB	bach_wtc2f20	Poly	733	3.00	9.00	0.35	0.75	0.48	0.66	0.71	0.68	0.31	0.69	0.43	80883	0.52	0.39	0.58	0.63	0.61			
COSIATECSegment	bach_wtc2f20	Poly	733	3.00	9.00	0.33	0.65	0.44	0.48	0.40	0.44	0.26	0.45	0.33	78887	0.63	0.34	0.43	0.48	0.45			
SIATECCompress	bach_wtc2f20	Poly	733	3.00	21.00	0.29	0.65	0.40	0.89	0.44	0.59	0.20	0.51	0.29	16124	0.17	0.25	0.53	0.42	0.47			
SIATECCompressBB	bach_wtc2f20	Poly	733	3.00	21.00	0.32	0.66	0.43	0.61	0.72	0.66	0.22	0.52	0.31	15627	0.22	0.26	0.47	0.58	0.52			
SIATECCompressSegment	bach_wtc2f20	Poly	733	3.00	21.00	0.31	0.66	0.43	0.47	0.35	0.40	0.20	0.43	0.27	16416	0.40	0.24	0.37	0.44	0.40			
COSIATEC	beet_sonata01-3	Mono	638	7.00	9.00	0.19	0.25	0.22				0.22	0.32	0.26	43945	0.22	0.25	0.49	0.49	0.49			
COSIATECBB	beet_sonata01-3	Mono	638	7.00	9.00	0.36	0.50	0.42	0.38	0.76	0.50	0.39	0.54	0.45	47563	0.48	0.50	0.72	0.68	0.70			
COSIATECSegment	beet_sonata01-3	Mono	638	7.00	9.00	0.56	0.59	0.58	0.68	0.81	0.74	0.59	0.61	0.60	44723	0.55	0.64	0.73	0.76	0.75			
SIATECCompress	beet_sonata01-3	Mono	638	7.00	24.00	0.15	0.26	0.19				0.17	0.30	0.22	9169	0.20	0.27	0.42	0.64	0.51			
SIATECCompressBB	beet_sonata01-3	Mono	638	7.00	24.00	0.50	0.68	0.57	0.30	0.76	0.43	0.44	0.63	0.52	9524	0.49	0.58	0.40	0.67	0.50			
SIATECCompressSegment	beet_sonata01-3	Mono	638	7.00	24.00	0.64	0.85	0.73	0.49	0.86	0.62	0.55	0.70	0.62	10003	0.54	0.71	0.44	0.86	0.58			
COSIATEC	beet_sonata01-3	Poly	1542	7.00	13.00	0.08	0.11	0.09				0.08	0.12	0.09	268193	0.11	0.10						
COSIATECBB	beet_sonata01-3	Poly	1542	7.00	13.00	0.32	0.43	0.37				0.31	0.46	0.37	263420	0.30	0.30	0.56	0.70	0.62			
COSIATECSegment	beet_sonata01-3	Poly	1542	7.00	13.00	0.49	0.59	0.53	0.72	0.83	0.77	0.48	0.59	0.53	266858	0.32	0.37	0.72	0.78	0.75			
SIATECCompress	beet_sonata01-3	Poly	1542	7.00	30.00	0.16	0.24	0.19				0.14	0.24	0.18	50471	0.20	0.19	0.25	0.31	0.28			
SIATECCompressBB	beet_sonata01-3	Poly	1542	7.00	30.00	0.54	0.77	0.63	0.48	0.89	0.63	0.44	0.67	0.53	50242	0.42	0.36	0.44	0.83	0.58			
SIATECCompressSegment	beet_sonata01-3	Poly	1542	7.00	30.00	0.71	0.88	0.78	0.48	0.96	0.64	0.58	0.74	0.65	51634	0.55	0.49	0.48	0.91	0.63			
COSIATEC	chop_mazurka24-4	Mono	747	4.00	13.00	0.11	0.43	0.18	0.82	0.82	0.82	0.11	0.49	0.18	203570	0.43	0.21	0.82	0.71	0.76			
COSIATECBB	chop_mazurka24-4	Mono	747	4.00	13.00	0.21	0.58	0.31	0.85	0.85	0.85	0.21	0.61	0.31	176605	0.50	0.27	0.85	0.71	0.77			
COSIATECSegment	chop_mazurka24-4	Mono	747	4.00	13.00	0.38	0.79	0.51	0.65	0.75	0.70	0.35	0.74	0.47	210537	0.73	0.42	0.63	0.72	0.67			
SIATECCompress	chop_mazurka24-4	Mono	747	4.00	20.00	0.14	0.45	0.21	0.82	0.82	0.82	0.13	0.48	0.20	33806	0.43	0.28	0.82	0.71	0.76			
SIATECCompressBB	chop_mazurka24-4	Mono	747	4.00	20.00	0.30	0.61	0.41	0.85	0.85	0.85	0.25	0.61	0.35	34233	0.50	0.33	0.64	0.73	0.68			
SIATECCompressSegment	chop_mazurka24-4	Mono	747	4.00	20.00	0.48	0.78	0.59	0.41	0.81	0.55	0.41	0.71	0.52	34816	0.60	0.47	0.52	0.79	0.63			
COSIATEC	chop_mazurka24-4	Poly	2079	3.00	23.00	0.04	0.17	0.07				0.05	0.26	0.09	5757114	0.12	0.09						
COSIATECBB	chop_mazurka24-4	Poly	2079	3.00	23.00	0.14	0.52	0.22				0.15	0.61	0.24	5714344	0.32	0.22	0.67	0.73	0.70			
COSIATECSegment	chop_mazurka24-4	Poly	2079	3.00	23.00	0.34	0.74	0.47	0.63	0.90	0.74	0.32	0.78	0.45	5200779	0.65	0.47	0.55	0.90	0.69	0.04	0.33	0.08
SIATECCompress	chop_mazurka24-4	Poly	2079	3.00	45.00	0.06	0.19	0.10				0.07	0.25	0.11	309682	0.13	0.27						
SIATECCompressBB	chop_mazurka24-4	Poly	2079	3.00	45.00	0.30	0.58	0.40				0.25	0.50	0.33	313249	0.39	0.51	0.41	0.62	0.50			
SIATECCompressSegment	chop_mazurka24-4	Poly	2079	3.00	45.00	0.50	0.81	0.61	0.42	0.88	0.57	0.41	0.68	0.51	324339	0.46	0.63	0.35	0.84	0.50			
COSIATEC	gbns_silverswan	Mono	347	8.00	4.00	0.54	0.38	0.44	1.00	0.63	0.78	0.33	0.26	0.29	11159	0.38	0.33	0.70	0.59	0.64			
COSIATECBB	gbns_silverswan	Mono	347	8.00	4.00	0.52	0.53	0.52	1.00	0.63	0.78	0.37	0.32	0.34	11249	0.53	0.37	1.00	0.63	0.78			
COSIATECSegment	gbns_silverswan	Mono	347	8.00	4.00	0.72	0.81	0.76	0.35	0.73	0.48	0.43	0.37	0.40	10983	0.81	0.43	0.36	0.74	0.49			
SIATECCompress	gbns_silverswan	Mono	347	8.00	17.00	0.30	0.43	0.35				0.14	0.25	0.18	2413	0.30	0.19	0.21	0.43	0.28			
SIATECCompressBB	gbns_silverswan	Mono	347	8.00	17.00	0.41	0.56	0.48				0.30	0.29	0.30	2821	0.39	0.32	0.21	0.42	0.28			
SIATECCompressSegment	gbns_silverswan	Mono	347	8.00	17.00	0.47	0.78	0.59	0.23	0.74	0.35	0.34	0.34	0.34	2857	0.61	0.39	0.26	0.75	0.38			
COSIATEC	gbns_silverswan	Poly	347	4.00	7.00	0.31	0.43	0.36				0.17	0.28	0.21	6600	0.43	0.22	0.43	0.34	0.38			
COSIATECBB	gbns_silverswan	Poly	347	4.00	7.00	0.41	0.54	0.47				0.39	0.40	0.40	6811	0.54	0.46	0.38	0.50	0.44			
COSIATECSegment	gbns_silverswan	Poly	347	4.00	7.00	0.61	0.50	0.55	0.74	0.90	0.81	0.55	0.37	0.44	6753	0.50	0.63	0.58	0.78	0.66			
SIATECCompress	gbns_silverswan	Poly	347	4.00	17.00	0.27	0.45	0.33				0.13	0.23	0.16	1957	0.23	0.12	0.20	0.24	0.22			
SIATECCompressBB	gbns_silverswan	Poly	347	4.00	17.00	0.33	0.43	0.38				0.31	0.32	0.32	2471	0.33	0.36	0.26	0.39	0.31			
SIATECCompressSegment	gbns_silverswan	Poly	347	4.00	17.00	0.46	0.37	0.41	0.63	0.83	0.72	0.49	0.34	0.40	2474	0.32	0.61	0.56	0.83	0.67			
COSIATEC	mzrt_sonata04-2	Mono	923	9.00	12.00	0.28	0.42	0.34	0.94	0.94	0.94	0.31	0.47	0.37	301756	0.22	0.33	0.96	0.79	0.87	0.08	0.11	0.10
COSIATECBB	mzrt_sonata04-2	Mono	923	9.00	12.00	0.36	0.48	0.41	0.92	0.92	0.92	0.37	0.51	0.43	304746	0.23	0.34	0.94	0.78	0.85	0.08	0.11	0.10
COSIATECSegment	mzrt_sonata04-2	Mono	923	9.00	12.00	0.42	0.56	0.48	0.88	0.79	0.84	0.43	0.56	0.48	302614	0.24	0.35	0.91	0.72	0.80	0.08	0.11	0.10
SIATECCompress	mzrt_sonata04-2	Mono	923	9.00	22.00	0.21	0.33	0.26				0.18	0.32	0.23	41625	0.24	0.29	0.40	0.44	0.42			
SIATECCompressBB	mzrt_sonata04-2	Mono	923	9.00	22.00	0.40	0.59	0.48	0.55	0.74	0.63	0.36	0.52	0.43	40290	0.34	0.37	0.40	0.65	0.49			
SIATECCompressSegment	mzrt_sonata04-2	Mono	923	9.00	22.00	0.57	0.73	0.64	0.55	0.85	0.67	0.50	0.60	0.54	42124	0.41	0.40	0.55	0.79	0.65			
COSIATEC	mzrt_sonata04-2	Poly	1744	9.00	18.00	0.15	0.32	0.21				0.17	0.36	0.23	1735091	0.32	0.44	0.50	0.52	0.51			
COSIATECBB	mzrt_sonata04-2	Poly	1744	9.00	18.00	0.28	0.47	0.35	0.65	0.85	0.74	0.28	0.46	0.34	1783759	0.46	0.57	0.49	0.62	0.54			
COSIATECSegment	mzrt_sonata04-2	Poly	1744	9.00	18.00	0.41	0.71	0.52	0.70	0.71	0.70	0.41	0.53	0.46	1744615	0.54	0.61	0.57	0.64	0.61	0.06	0.11	0.07
SIATECCompress	mzrt_sonata04-2	Poly	1744	9.00	33.00	0.17	0.33	0.22				0.18	0.35	0.23	162704	0.21	0.40	0.54	0.42	0.47			
SIATECCompressBB	mzrt_sonata04-2	Poly	1744	9.00	33.00	0.48	0.66	0.56	0.56	0.84	0.67	0.44	0.58	0.50	157833	0.34	0.57	0.47	0.78	0.59			
SIATECCompressSegment	mzrt_sonata04-2	Poly	1744	9.00	33.00	0.56	0.69	0.62	0.50	0.89	0.64	0.54	0.60	0.57	164372	0.37	0.61	0.46	0.86	0.60			

Table 1. Results of running COSIATEC and SIATECCOMPRESS on the JKU Patterns Development Database.

precision scores tend to be more similar to each other.

2. The algorithms score higher on occurrence measures than establishment measures.

3. On "three-layer" measures (P_3, R_3 and TLF_1), the algorithms generally score better on recall than precision.

4. SIATECCOMPRESS is 5–10 times faster than COSIATEC and clearly has a lower order of growth with respect to input size. A more detailed analysis of runtime will be given in a later paper.

5. The highest establishment F_1 score of 0.78 was obtained using SIATECCompressSegment on the Beethoven Sonata movement.

6. The highest occurrence F_1 score with $c = 0.75$ of 0.94 was obtained using COSIATEC on the monophonic Mozart Sonata movement. On this movement, the algorithm also achieved occurrence precision and occurrence recall of 0.94.

7. The highest values of the "three-layer" F_1 score (0.62–0.65) were obtained using SIATECCompressSegment on the Beethoven Sonata movement (similar values were obtained for both the monophonic and polyphonic versions).

8. The highest values of the occurrence F_1 score with $c = 0.5$ were 0.85–0.87 obtained using COSIATEC and COSIATECBB on the monophonic version of the Mozart Sonata movement.

8. CONCLUSIONS

The results indicate that the output of COSIATEC and SIATECCOMPRESS is clearly related to the human-identified patterns annotated in the JKU PDD ground truth. However, evaluating a musical analysis algorithm by how well its output predicts whether or not a pattern is considered "important" or "interesting" by some particular analyst seems somewhat arbitrary. The goal of music analysis is to find the best ways of understanding musical works—that is, those ways that allow us to more effectively carry out expert musical tasks. Such tasks could include, for example, identifying errors in scores or performances, correctly identifying authorship or completing partial compositions. Simply claiming that a pattern is a "pattern of interest" or "perceptually salient" or "structurally important" doesn't really mean very much, unless one can show how knowing about the pattern helps with carrying out some expert musical task more effectively. Nevertheless, the first MIREX competition on Pattern Discovery is an important step towards the development of rigorous methodologies for evaluating algorithms for musical pattern discovery.

9. REFERENCES

[1] Tom Collins. *Improved methods for pattern discovery in music, with applications in automated stylistic composition*. PhD thesis, Faculty of Mathematics, Computing and Technology, The Open University, Milton Keynes, 2011.

[2] Tom Collins. Mirex 2013 competition: Discovery of repeated themes and sections, 2013. `http://www.music-ir.org/mirex/wiki/2013:Discovery_of_Repeated_Themes_&_Sections`. Accessed on 7 October 2013.

[3] James C. Forth. *Cognitively-Motivated Geometric Methods of Pattern Discovery and Models of Similarity in Music*. PhD thesis, Department of Computing, Goldsmiths, University of London, 2012.

[4] Jamie Forth and Geraint A. Wiggins. An approach for identifying salient repetition in multidimensional representations of polyphonic music. In J. Chan, J. W. Daykin, and M. S. Rahman, editors, *London Algorithmics 2008: Theory and Practice*, pages 44–58. College Publications, London, 2009.

[5] David Meredith. Point-set algorithms for pattern discovery and pattern matching in music. In *Proceedings of the Dagstuhl Seminar on Content-based Retrieval (No. 06171, 23–28 April, 2006)*, Schloss Dagstuhl, Germany, 2006. Available online at <http://drops.dagstuhl.de/opus/volltexte/2006/652>.

[6] David Meredith. The *ps13* pitch spelling algorithm. *Journal of New Music Research*, 35(2):121–159, 2006.

[7] David Meredith. *Computing Pitch Names in Tonal Music: A Comparative Analysis of Pitch Spelling Algorithms*. PhD thesis, Faculty of Music, University of Oxford, 2007.

[8] David Meredith, Kjell Lemström, and Geraint A. Wiggins. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*, 31(4):321–345, 2002.

[9] David Meredith, Kjell Lemström, and Geraint A. Wiggins. Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. In *Cambridge Music Processing Colloquium*, 2003.