USING SIATECCOMPRESS
TO DISCOVER REPEATED THEMES AND SECTIONS
IN POLYPHONIC MUSIC

David Meredith
Aalborg University
dave@titanmusic.com

ABSTRACT

Three variants of the SIATECCOMPRESS algorithm were submitted to the 2016 MIREX competition on Discovery of Repeated Themes and Sections. The three variants were those that performed best in terms of three-layer $F_1$ score (TLF1), three-layer precision (TLP) and three-layer recall (TLR), respectively, when a large number of variants of SIATECCOMPRESS, COSIATEC and Forth’s algorithm were run on the polyphonic version of the JKU Patterns Development Database (JKU-PDD). The variant optimised for TLF1 achieved an average TLF1 score of 0.490 over the five pieces in the database. The variant optimised for TLP achieved an average TLP score of 0.561 over the JKU-PDD. The variant optimised for TLR achieved an average TLR score of 0.583 over the JKU-PDD. The three variants have been implemented in Java.

1. INTRODUCTION

COSIATEC [8,11–16,18,19], SIATECCOMPRESS [12–16], and Forth’s algorithm [6,7] are greedy point-set compression algorithms that have been developed for analysing music. All three algorithms are based on the SIA and SIATEC algorithm described by Meredith, Lemström and Wiggins [17]. 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 and Forth’s algorithm may share points (i.e., notes).

All three 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.

In previous experiments [12,15,16], it was found that, of the three algorithms, versions of SIATECCOMPRESS performed best on the task of discovering repeated themes and sections in both the JKU Patterns Development Database (JKU-PDD) [2] and the test database used in the 2013 and 2014 MIREX competitions [3,4]. Three variants of the SIATECCOMPRESS algorithm were therefore submitted to the 2016 competition, optimized for precision, recall and $F_1$ score, respectively.

2. USING POINT SETS TO REPRESENT MUSIC

The submitted algorithms 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. [17]. Although these algorithms work with datasets of any dimensionality, it will be assumed here that each dataset is a set of two-dimensional points, $(t,p)$, where each point represents a single note or sequence of tied notes whose onset time is $t$ in tattums and whose morphetic pitch [9,10,17] 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 [9,10].

The three variants of SIATECCOMPRESS submitted to the 2016 MIREX competition on Discovery of Repeated Themes and Sections take Collins’ “lisp” format files as input and use morphetic pitch representation.

3. MAXIMAL TRANSLATABLE PATTERNS

I shall use the term pattern to refer to any subset of a dataset. Suppose $D$ is a dataset and $D_1', D_2' \subseteq D$. The two patterns, $D_1', D_2'$, are said to be translationally equivalent, denoted by $D_1' \equiv_T D_2'$, if and only if there exists a vector $v$, such that $D_1'$ translated by $v$ is equal to $D_2'$. That is,

$$D_1' \equiv_T D_2' \iff (\exists v \mid D_2' = D_1' + v). \quad (1)$$

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

$$\text{MTP}(v, D) = \{ p \mid p \in D \land p + v \in D \} \quad (2)$$

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 $D$. 

This document is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 License. http://creativecommons.org/licenses/by-nc-sa/3.0/
© 2013 David Meredith.
The notion that COSIATEC, SIATECCOMPRESS and Forth’s algorithm 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. [17] 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, \( D' \), in a dataset, \( D \), the translational equivalence class (TEC) of \( D' \) in \( D \) is defined and denoted as follows:

\[
\text{TEC}(D', D) = \{ Q \mid Q \equiv_T P \land Q \subseteq D \} .
\]  

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

\[
\text{COV}(T) = \bigcup_{D' \in T} \text{D}' .
\]

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. [17] 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, \( D' \), and then give the \( n-1 \) vectors required to map \( D' \) 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, \( \langle D', V \rangle \), where \( D' \) is one occurrence in \( T \) and \( V \) is the set of vectors that map \( D' \) onto the other occurrences in \( T \). When a TEC, \( T = \langle D', V \rangle \), is represented in this way, we call \( D' \) the pattern and \( V \) the translator set of the TEC.

5. THE COSIATEC ALGORITHM

COSIATEC [8, 18] (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, \( T \), such that

\[
D = \bigcup_{T \in T} \text{COV}(T)
\]

and, for all \( T_1, T_2 \in T \), \( T_1 \neq T_2 \),

\[
\text{COV}(T_1) \cap \text{COV}(T_2) = \emptyset .
\]

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 (pattern, translator set) 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 \( D' \) (line 1). Then, on each iteration of the while loop (lines 4–7), the algorithm finds the “best” MTP TEC in \( D' \), \( T^* \), appends this TEC to \( T \) and then removes the set of points covered by \( T^* \) from \( D' \). When \( D' \) is empty, the algorithm terminates, returning the list of MTP TECs, \( T \). 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.

Given an input dataset, \( D \), and what remains of a copy, \( D' \), of this dataset after the removal of zero or more MTP TEC covered sets, the COSIATEC algorithm finds the “best” MTP TEC in \( D' \) (line 5), using the GetBestTEC algorithm shown in Figure 2. In lines 1–2 of GetBestTEC, 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, \( V \), which is a two-dimensional array of ordered triples,

\[
V[i][j] = (p_i - p_j, p_j, j) ,
\]

where \( p_i - p_j \) is the vector from point \( p_j \) to \( p_i \), and \( p_k = D'[k] \), where \( D' \) is an ordered set that only contains every element in \( D' \), sorted into lexicographical order.

Having computed the vector table, \( V \), the MTPs are found by sorting the triples in \( V \), lexicographically by their vectors (i.e., their first elements), and then scanning this
GetBestTEC(D', D)
1 V ← ComputeVectorTable(D')
2 MCPs ← ComputeMtpCIsPairs(V)
3 mcp ← nil
4 T* ← nil
5 for i ← 0 to |MCPs| - 1
6 mcp ← MCPs[i]
7 T ← GETTECForMTP(mcp, V, D')
8 T* ← GETCONJUGATETEC(T)
9 T ← REMOVEREDUNDANTTRANSLATORS(T)
10 T′ ← REMOVEREDUNDANTTRANSLATORS(T')
11 if T* = nil ∨ ISBETTERTEC(T, T*)
12 T ← T′
13 M ← T* + T′
14 return T*

Figure 2. The GetBestTEC algorithm.

ComputeMtpCIsPairs(V)
1 W ← SortByVectorTable(V)
2 MCPs ← {}  
3 CISs ← {}  
4 mtp ← (W[0][0])
5 mtp ← (W[0][1])
6 cis ← (W[0][2])
7 for i ← 1 to |W| - 1
8 vpi ← W[i]
9 if vpi = mtp
10 mtp ← mtp ∪ vpi[1]
11 cis ← cis ∪ vpi[2]
12 else
13 MCPs ← MCPs ∪ mtp
14 CISs ← CISs ∪ cis
15 mtp ← vpi[1]
16 cis ← vpi[2]
17 v ← vpi[0]
18 MCPs ← MCPs ∪ mtp
19 CISs ← CISs ∪ cis
20 MCPs ← {}
21 for i ← 0 to |MCPs| - 1
22 MCPs ← MCPs ∪ ⟨MTPs[i], CISs[i]⟩
23 return MCPs

Figure 3. The ComputeMtpCIsPairs algorithm.

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.

The ComputeMtpCIsPairs algorithm (Figure 3) first sorts the triples in the vector table, V, into increasing lexicographical order by their vectors. The resulting ordered set of triples is stored in the variable W (see line 1). In lines 2–19 of this algorithm, two lists are constructed, MCPs and CISs. MCPs contains all the MTPs in the dataset, each MTP being represented as an ordered set of points in lexicographical order. 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 ⟨mtp, cis⟩ pairs is constructed by combining corresponding elements in MCPs and CISs.

In lines 5–14 of GetBestTEC, the for loop iterates over this ordered set of ⟨mtp, cis⟩ pairs computed by

\[
\text{CR}(T) = \frac{|\text{COV}(T)|}{|P(T)| + |V(T)| - 1}.
\]
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) [17]. 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 $O(n^3)$ where $n$ is the number of points in the input dataset, running COSIATEC on large datasets can be time-consuming (see Table ?? 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 [6, 7], 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, MCPs, that are translationally equivalent to MTPs that occur earlier in this list. This eliminates the possibility of the same TEC 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, $T$. In line 6, redundant translators are removed from the translator set of each TEC in $T$ 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.

6.1 Forth’s algorithm

Forth [6, 7] presents an algorithm, that, like COSIATEC, computes a set of TECs that collectively cover the input dataset. However, unlike COSIATEC, Forth’s algorithm runs SIATEC only once and the covers it generates are not, in general, strict partitions—in the output of Forth’s algorithm, the TECs may share covered points and, collectively, may not completely cover the input dataset.

The first step in Forth’s algorithm is to run SIATEC on the input dataset, $D$. This generates a sequence of MTP TECs, $T = \langle T_1, T_2, \ldots, T_n \rangle$. The algorithm then computes the covered set, $C_i = \text{COV}(T_i)$, for each TEC,
$T_i$, in $\mathbf{T}$, to produce a sequence of TEC covered sets, $\mathbf{C} = \langle C_1, C_2, \ldots, C_n \rangle$. It then assigns a weight, $W_i$, to each covered set, $C_i$, to produce the sequence, $\mathbf{W} = \langle W_1, W_2, \ldots, W_n \rangle$. $W_i$ is intended to measure the “structural salience” [6, p. 41] of the patterns in $T_i$, and it is defined as follows:

$$W_i = w_{cr,i} \cdot w_{\text{compV},i},$$

where $w_{cr,i}$ is a normalized value representing the compression ratio of $T_i$ and $w_{\text{compV},i}$ is a normalized value representing the within-voice segment compactness of the patterns in $T_i$. Forth [6, p. 38] defines the compression ratio of a TEC in the same way as Meredith et al. [18].

The within-voice segment compactness of a pattern, as used in Forth’s algorithm, is typically approximately equal to its bounding-box compactness (as used in COSIATEC). However, an advantage of bounding-box compactness over within-voice segment compactness is that the former does not depend on information about voice structure being present in the input data—that is, bounding-box compactness does not require the musical surface to have already been parsed into unambiguous voices. This avoids the need for selecting one, unambiguous voice structure interpretation for a piece in cases where the voice structure is actually ambiguous (as it often is, for example, in keyboard music). Nevertheless, if information about voice structure is available, one might well hypothesize that within-voice segment compactness would give better results than bounding-box compactness (though the results reported in section ?? do not support this hypothesis).

Having computed the sequence of weights, $\mathbf{W}$, Forth’s algorithm then attempts to select a subset of $\mathbf{C}$ that covers the input dataset while maximising the weights, $W_i$, of the TECs used in the cover. Forth’s algorithm takes $D$, $\mathbf{C}$ and $\mathbf{W}$ as parameters, along with a numerical parameter, $c_{\text{min}}$. The algorithm repeatedly selects the “best” remaining TEC covered set, $C^*$, in $\mathbf{C}$, adds this to the cover, $\mathbf{S}$, and then removes $C^*$ from $\mathbf{C}$. A point set, $P$, is used to store the set of points covered by the TEC covered sets selected so far. In order for a TEC covered set to be added to the cover, the number of new points that it covers, $c$, (i.e., that are not already in $P$) must be at least $c_{\text{min}}$. The TEC covered set, $C^*$, that is added on a particular iteration is then the one for which $c \cdot W_i$ is a maximum. If no TEC covered set is selected on a particular iteration, then the algorithm terminates, even if the dataset has not been completely covered. The algorithm returns the cover, $\mathbf{S}$, containing the “best” TECs selected.

7. EVALUATION

Two experiments were carried out in order to select the specific variants of the algorithms to be submitted to the 2016 MIREX competition. In the first experiment, 96 variants of each of the three basic algorithms (i.e., COSIATEC, SIATECCOMPRESS and Forth’s algorithm) were run on the polyphonic version of the JKU-PDD [2]. For each of these three basic algorithms, the following parameter values were used:

1. compactness trawling [5] was either used or not used;
2. SIA was either replaced with SIAR [1] or it was not;
3. redundant translators were either removed or not;
4. minimum permitted compactness of patterns was set to either 0.5 or 0.9;
5. minimum pattern size was set to either 4 or 8;
6. the algorithms were run in either raw, bb or segment mode.

The morphetic pitch representations of the polyphonic versions of the JKU-PDD were used and the algorithms were run on the “lisp” encodings of the pieces.

The performance of the 288 algorithm variants was measured using using Tom Collins’ MATLAB implementation of the metrics defined in [3], bundled with the JKU PDD. Particular attention was paid to the values of three-layer precision, three-layer recall and three-layer $F_1$ measure [15].

The algorithm variants that performed best in this experiment in terms of three-layer $F_1$ score were versions of SIATECCOMPRESS with the following parameter values:

1. compactness trawling was not used;
2. SIAR was used in place of SIA;
3. redundant translators were not removed;
4. minimum pattern size was set to 4;
5. the algorithms were run in segment mode.

For these parameter values, the choice of minimum compactness threshold made no difference to the $F_1$ score. With these parameter values, the three-layer $F_1$ scores achieved over the five pieces in the database were as follows:

- Bach: 0.27952
- Beethoven: 0.63336
- Chopin: 0.60895
- Gibbons: 0.36589
- Mozart: 0.5609
- Mean $F_1$ score over all five pieces: 0.489724

Table 1 shows the values of three-layer precision for the algorithms that performed best on average over the JKU-PDD in terms of this measure. Note that the top 8 algorithms were versions of either Forth’s algorithm or SIATECCOMPRESS, and that the parameter values in these algorithms were as follows:

1. SIAR was used instead of SIA (indicated by “rd” in the algorithm name);
2. minimum pattern size was set to 8 (indicated by “min8” in the algorithm name);

3. the algorithms were run in segment mode (indicated by “segmode” in the algorithm name).

Whether redundant translators were removed or not and the choice of minimum compactness made no difference in terms of precision.

The algorithm variants that performed best in this first experiment in terms of three-layer recall were the versions of SIATECCOMPRESS with the following parameter values:

1. compactness trawling was not used;
2. SIA was used instead of SIAR;
3. redundant translators were not removed;
4. minimum pattern size was set to 4;
5. segment mode was used.

For these algorithms, the minimum compactness setting made no difference to the precision.

This first experiment confirmed previous results that showed that SIATECCOMPRESS appears to be the most successful of the three basic algorithms on this task. A second, more in-depth experiment was therefore carried out to find the variants of SIATECCOMPRESS that perform best on the JKU-PDD in terms of three-layer precision, recall and $F_1$ score. In this second experiment, 72 variants of SIATECCOMPRESS were run in segment mode on the JKU-PDD, using the following parameter values:

1. either SIA or SIAR was used; if SIAR was used, the $r$ parameter was set to either 1 or 3;
2. redundant translators were either removed or not;
3. minimum compactness was set to either 0.25 or 0.5 (minimum compactness settings has made no difference to the results in the first experiment);
4. minimum pattern size was set to 4, 6 or 8;
5. the number of patterns generated by the algorithms was either unlimited or limited to 10 patterns.

Tables 2 to 4 show, respectively, the three-layer $F_1$, precision and recall scores for those variants of SIATECCOMPRESS tested that performed best in terms of these measures.

The results in Table 2 indicate that SIATECCOMPRESS performed best in terms of three-layer $F_1$ score over the JKU-PDD when

- SIAR was used with $r$ set to 1;
- minimum pattern size was set to 4;
- the number of returned patterns was unlimited (“top0”).

Removing redundant translators only very marginally reduced performance. Minimum compactness threshold made no difference to these results. The two best variants achieved a three-layer $F_1$ score of 0.4897. The variant submitted to the MIREX competition as algorithm MeredithTLF1MIREX2016.jar was therefore a version of SIATECCOMPRESS with the following parameter values:

- SIAR used with $r = 1$;
- minimum compactness set to 0.25;
- minimum pattern size set to 4;
- number of patterns generated unlimited;
- segment mode used;
- redundant translators not removed;
- morphetic pitch representation used.

The results in Table 3 indicate that SIATECCOMPRESS performed best in terms of three-layer precision over the JKU-PDD when

- SIAR was used with $r = 1$;
- redundant translators were not removed;
- minimum pattern size was set to 8;
- the number of returned patterns was limited to 10.

Minimum compactness threshold made no difference to these results. The two best variants achieved a three-layer precision of 0.5614. The variant submitted to the MIREX competition as algorithm MeredithTLPMIREX2016.jar
Table 2. Top-scoring variants of SIATECCOMPRESS in the second experiment in terms of three-layer $F_1$ score over the JKU-PDD.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Bach</th>
<th>Beet</th>
<th>Chop</th>
<th>Gbns</th>
<th>Mtrt</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r1-minc0.25-min4-bbcomp-segnode-top0</td>
<td>0.27952</td>
<td>0.63336</td>
<td>0.60895</td>
<td>0.36589</td>
<td>0.5609</td>
<td>0.489724</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r1-minc0.5-min4-bbcomp-segnode-top0</td>
<td>0.27952</td>
<td>0.63336</td>
<td>0.60895</td>
<td>0.36589</td>
<td>0.5609</td>
<td>0.489724</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r3-rrt-minc0.25-min4-bbcomp-segnode-top0</td>
<td>0.2697</td>
<td>0.61844</td>
<td>0.61895</td>
<td>0.38521</td>
<td>0.55818</td>
<td>0.489596</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r3-rrt-minc0.5-min4-bbcomp-segnode-top0</td>
<td>0.2697</td>
<td>0.61844</td>
<td>0.61895</td>
<td>0.38521</td>
<td>0.55818</td>
<td>0.489596</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-minc0.25-min4-bbcomp-segnode-top0</td>
<td>0.28792</td>
<td>0.65169</td>
<td>0.552</td>
<td>0.39973</td>
<td>0.54884</td>
<td>0.488036</td>
</tr>
</tbody>
</table>

Table 3. Top-scoring variants of SIATECCOMPRESS in the second experiment in terms of three-layer precision over the JKU-PDD.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Bach</th>
<th>Beet</th>
<th>Chop</th>
<th>Gbns</th>
<th>Mtrt</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r1-minc0.25-min8-bbcomp-segnode-top10</td>
<td>0.24153</td>
<td>0.6405</td>
<td>0.6147</td>
<td>0.67715</td>
<td>0.6363</td>
<td>0.561442</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r1-minc0.5-min8-bbcomp-segnode-top10</td>
<td>0.24153</td>
<td>0.6405</td>
<td>0.6147</td>
<td>0.67715</td>
<td>0.6363</td>
<td>0.561442</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r3-minc0.25-min8-bbcomp-segnode-top10</td>
<td>0.25614</td>
<td>0.60302</td>
<td>0.60138</td>
<td>0.67718</td>
<td>0.63356</td>
<td>0.553656</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r3-minc0.5-min8-bbcomp-segnode-top10</td>
<td>0.25614</td>
<td>0.60302</td>
<td>0.60138</td>
<td>0.67718</td>
<td>0.63356</td>
<td>0.553656</td>
</tr>
</tbody>
</table>

Table 4. Top-scoring variants of SIATECCOMPRESS in the second experiment in terms of three-layer recall over the JKU-PDD.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Bach</th>
<th>Beet</th>
<th>Chop</th>
<th>Gbns</th>
<th>Mtrt</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIATECCOMPRESS d-m-minc0.25-min4-bbcomp-segnode-top0</td>
<td>0.42892</td>
<td>0.74609</td>
<td>0.82127</td>
<td>0.34474</td>
<td>0.57459</td>
<td>0.583122</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-minc0.5-min4-bbcomp-segnode-top0</td>
<td>0.42892</td>
<td>0.74609</td>
<td>0.82127</td>
<td>0.34474</td>
<td>0.57459</td>
<td>0.583122</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rrt-minc0.25-min4-bbcomp-segnode-top0</td>
<td>0.42892</td>
<td>0.73008</td>
<td>0.83183</td>
<td>0.34474</td>
<td>0.56669</td>
<td>0.580452</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rrt-minc0.5-min4-bbcomp-segnode-top0</td>
<td>0.42892</td>
<td>0.73008</td>
<td>0.83183</td>
<td>0.34474</td>
<td>0.56669</td>
<td>0.580452</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r3-rrt-minc0.25-min4-bbcomp-segnode-top0</td>
<td>0.36738</td>
<td>0.68142</td>
<td>0.86171</td>
<td>0.35277</td>
<td>0.6106</td>
<td>0.574776</td>
</tr>
<tr>
<td>SIATECCOMPRESS d-m-rsd-r3-rrt-minc0.5-min4-bbcomp-segnode-top0</td>
<td>0.36738</td>
<td>0.68142</td>
<td>0.86171</td>
<td>0.35277</td>
<td>0.6106</td>
<td>0.574776</td>
</tr>
</tbody>
</table>
was therefore a version of SIATECCOMPRESS with the same parameter values as MeredithTLF1MIREX2016.jar, except the minimum pattern size was increased to 8 and the number of patterns generated was limited to 10.

The results in Table 4 indicate that SIATECCOMPRESS performed best in terms of three-layer recall over the JKUPDD when

- SIA was used instead of SIAR;
- redundant translators were not removed;
- minimum pattern size was set to 4;
- the number of patterns generated was unlimited.

Minimum compactness threshold made no difference to recall. The two best variants achieved a three-layer recall value of 0.5831. The variant submitted to the MIREX competition as algorithm MeredithTLRMIREX2016.jar was therefore a version of SIATECCOMPRESS in segment mode in which

- SIA was used instead of SIAR;
- redundant translators were not removed;
- minimum pattern size was set to 4;
- the number of patterns generated was unlimited;
- minimum compactness was set to 0.25.

8. REFERENCES


