COMPRESSION-BASED GEOMETRIC PATTERN DISCOVERY IN MUSIC

David Meredith

Aalborg University
Department of Architecture, Design and Media Technology
Sofiendalsvej 11, 9200 Aalborg SV, Denmark
dave@create.aau.dk

ABSTRACT

The purpose of musical analysis is to find the best possible explanations for musical objects, where such objects may range from single chords or phrases to entire musical corpora. Kolmogorov complexity theory suggests that the best possible explanation for an object is represented by the shortest possible description of it. Two compression algorithms, COSIATEC and SIATECCOMPRESS, are described that take point-set representations of musical objects as input and generate compressed encodings of these point sets as output. The algorithms were evaluated on a task in which 360 folk songs were classified into tune families using normalized compression distance, a 1-nn classifier and leave-one-out cross-validation. COSIATEC achieved a success rate of 84% on this task, compared with a success rate of 13% for a general-purpose compressor. Variants of the algorithms incorporating modifications that have been suggested in the literature were also run on the task and the results were compared.

Index Terms— Pattern discovery, Compression, Music information retrieval, Music analysis, Machine learning

1. INTRODUCTION

A *musical analysis* represents a particular way of understanding a *musical object*, where such an object could be any quantity of music, ranging from a single chord to an entire corpus. In music theory, analyses have traditionally been evaluated subjectively, in terms of how satisfactorily an analysis is perceived to account for the structure of the musical object in question. However, whether or not an analysis improves one's understanding of a musical object can also be determined by whether it allows for certain objectively evaluable tasks to be performed more successfully. For example, a good analysis of a piece might be expected to help with tasks such as error detection, authorship identification or memorisation of a piece. One analysis of a musical object could therefore be considered "better" than another if it allows such tasks to be performed more successfully. It thus makes sense to propose that the goal of music analysis is to find the best possible ways of understanding musical objects.

The problem addressed here is that of designing an algorithm that automatically generates high-quality analyses of musical objects from *in extenso* descriptions of these objects. The analyses produced by such an algorithm could be used in various practical, entertaining, educational, musicological and/or scientific applications such as, for example, indexing a database of music encodings for a content-based music search engine, composing new examples of music in an existing style, identifying the composers of unattributed works or developing engaging and educational musical games. Such an analysis algorithm could also provide a computational model of the processes that underlie certain aspects of human music perception and cognition.

The work presented here is based on the assumption that a musical analysis can be conceived of as an encoding in the form of a program that, when executed, generates as its output an *in extenso* description of a musical object. An *in extenso* description is one in which the properties of each *atomic component* of an object (e.g., a MIDI event, a note in a score or a sample in a PCM audio file) are explicitly specified, without encoding any structural groupings of these components into higher-level constituents, and without encoding any relationships between atomic components. For example, in an *in extenso* description of a score, one might simply list the notes, giving the pitch, duration, onset time, etc. of each one. In contrast, while a musical analysis is itself a description of a musical object, it is not usually *in extenso*, because it typically represents groupings of atomic components (such as notes or audio samples) into larger-scale constituents (such as motives, phrases, chords, voices, sections, etc.), along with relationships between these components or constituents (e.g., repetition, transposition, truncation, inversion, diminution, etc.). If such an analysis is represented by a program that outputs an *in extenso* description of a musical object, then such a program embodies an explanation for certain aspects of the structure of a musical object.

Suppose X and Y are two constituents of a musical object (e.g., two entries of the subject in a fugue) and that the

978-1-4799-1180-6/14/$31.00 ©2014 IEEE

transformation, T, maps X onto Y (e.g., T could be "shift in time by x crotchets and transpose by y semitones"). If T can be described more parsimoniously than Y, then the part of the musical surface consisting of X and Y (i.e, $X \cup Y$, or the union of the atomic components in X and Y) can be described more parsimoniously by giving an *in extenso* description of X together with a description of T, than it can by giving *in extenso* descriptions of both X and Y. In this way, by identifying structural relationships between constituents of a musical object, a musical analysis typically conveys (at least) as much information about that object as an *in extenso* description on the same level of detail, but does so more parsimoniously. In other words, a musical analysis is typically a *compact description* or *compressed encoding* of a musical object.

Kolmogorov complexity theory [1] suggests that the *length* of a program, whose output is an *in extenso* description of an object, can be used as a measure of the complexity of its corresponding explanation for the structure of that object: if we have two programs in the same programming language that generate the same output, then the shorter of the two will (usually) represent the simpler explanation for that output. The level of structural detail on which an analysis (encoded as a program) explains the structure of a musical object is determined by the granularity of the *in extenso* description that it generates. Typically, much of the detailed structure of an object will not be encoded in the program's output and will therefore go unexplained. The music analyst's goal to find the best possible explanations for the structures of musical objects therefore translates into the goal of finding the shortest possible "programs" (i.e., encodings) that generate the most detailed representations of as much music as possible. Or, to put it another way, the analyst's goal is to compress as much information as possible about as much music as possible into as short an encoding as possible.

With this goal in mind, the purpose of this paper is to present and evaluate two compression algorithms, COSIATEC and SIATECCOMPRESS, that take *in extenso* descriptions of musical objects as input and generate losslessly compressed encodings of these descriptions as output. The encodings generated by COSIATEC and SIATECCOMPRESS can themselves be understood to be compact "programs" that generate the *in extenso* descriptions given as input to these compression algorithms. Both algorithms achieve compression by identifying sets of occurrences of maximal repeated patterns in the data and encoding these sets of occurrences by encoding one occurrence explicitly along with the transformations that map that occurrence onto its other occurrences. COSIATEC and SIATECCOMPRESS use different greedy strategies to search for the set of maximal repeated patterns that allow for the most compact encoding of the input data.

COSIATEC strictly partitions the input dataset into occurrence sets of maximal repeated patterns. However, the occurrences used by SIATECCOMPRESS in its encodings may overlap and share datapoints, resulting in less compact encodings of the input data. On the other hand, SIATECCOMPRESS has a better worst-case running time than COSIATEC and can therefore be significantly faster than it on larger datasets.

COSIATEC and SIATECCOMPRESS are pattern discovery algorithms: their purpose is to compute as short a description as possible of a set of points in terms of a set of patterns and these patterns' occurrences. They aim to discover the patterns in a dataset that allow it to be described as parsimoniously as possible. If one equates brevity of description with quality of explanation, then one might say that the algorithms attempt to find the most "explanatory" patterns in a dataset.

In the next section, some fundamental concepts will be reviewed relating to the use of point sets to represent music. Previous work on the development of point-set pattern discovery algorithms for music will then be briefly reviewed. The COSIATEC and SIATECCOMPRESS algorithms will then be described. The results obtained when the algorithms were used to carry out a classification task will then be presented and discussed.

2. USING POINT SETS TO REPRESENT MUSIC

In COSIATEC and SIATECCOMPRESS, it is assumed that the music to be analysed is represented as a multi-dimensional point set called a *dataset* (as in [2]). 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 t and p are integers, representing, respectively, the onset time in tatums and the chromatic or diatonic pitch [2] of a note or sequence of tied notes in a musical score. Fig. 1 shows an example of such a dataset.

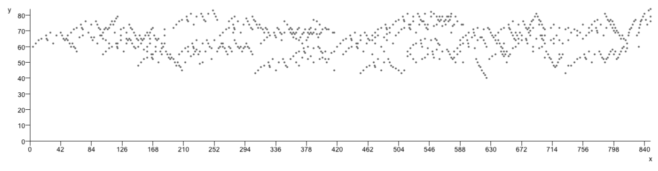


Fig. 1. A two-dimensional dataset representing the fugue from J. S. Bach's Prelude and Fugue in C minor, BWV846. The horizontal axis represents onset time in tatums; the vertical axis represents diatonic pitch. Each point represents a note or a sequence of tied notes.

2.1. Maximal translatable patterns (MTPs)

If D is a dataset, then any subset of D may be called a *pattern*. If $P_1, P_2 \subseteq D$, then 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, $P_1 \equiv_{\mathrm{T}} P_2 \iff (\exists v \mid P_2 = P_1 + v)$, where $P_1 + v$ denotes the pattern that results when P_1 is translated by the vector v. For example, in each of the graphs in Fig. 2, the pattern of circles is translationally equivalent to the pattern of crosses. A pattern, $P \subseteq D$, is said to be *translatable* within a dataset, D, if and only if there exists a vector, v, such that $P + v \subseteq D$. Given a vector, v, then the *maximal translatable pattern* (MTP) for v in the dataset, D, is given by $\mathrm{MTP}(v, D) = \{p \mid p \in D \wedge p+v \in D\}$, where $p+v$ is the point that results when p is translated 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. Fig. 2 shows some examples of maximal translatable patterns.

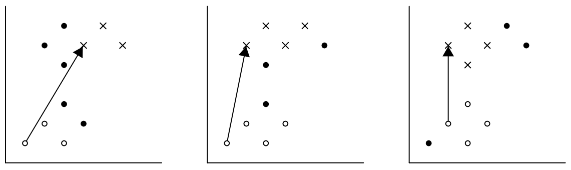


Fig. 2. Examples of maximal translatable patterns (MTPs). In each graph, the pattern of circles is the maximal translatable pattern (MTP) for the vector indicated by the arrow. The pattern of crosses in each graph is the pattern onto which the pattern of circles is mapped by the vector indicated by the arrow.

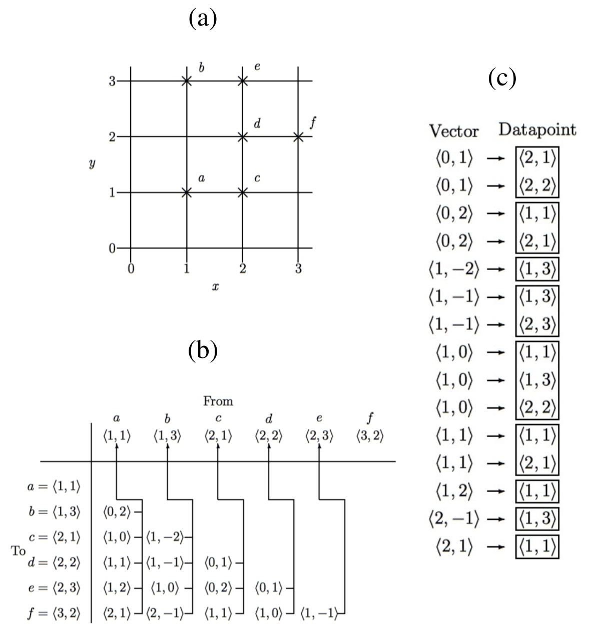
The development of COSIATEC and SIATECCOMPRESS was motivated by the hypothesis that the patterns that allow for the best possible interpretations of a piece are closely related to MTPs in the pitch-time point-set representation of the piece. Meredith *et al.* [2,3] describe an algorithm called SIA ("Structure Induction Algorithm") for discovering all the MTPs in a dataset. For a dataset of n, k–dimensional points, SIA has a worst-case running time of $O(kn^2 \log_2 n)$ and uses $O(kn^2)$ space. Fig. 3 describes the SIA algorithm.

2.2. Translational equivalence classes (TECs)

When analysing a piece of music, we typically want to find *all the occurrences* of an interesting pattern, not just one occurrence. Thus, if we believe that MTPs are related in some way to the patterns that listeners and analysts find interesting, then we want to be able to find all the occurrences of each MTP. Given a pattern, P, in a dataset, D, the *translational equivalence class* (TEC) of P in D is given by $\mathrm{TEC}(P, D) = \{Q \mid Q \equiv_{\mathrm{T}} P \wedge Q \subseteq D\}$. That is, the TEC of a pattern, P, in a dataset contains all and only those patterns in the dataset that are translationally equivalent to P. Note that $P \equiv_{\mathrm{T}} P$, so $P \in \mathrm{TEC}(P, D)$. Fig. 4 shows some examples of TECs.

We can also define the *covered set* of a TEC, T, denoted by $\mathrm{COV}(T)$, to be the union of the patterns in T. That is, $\mathrm{COV}(T) = \bigcup_{P \in T} P$. Here, we will be particularly concerned with *MTP TECs*—that is, the translational equivalence classes of the maximal translatable patterns in a dataset.

A TEC, $T = \mathrm{TEC}(P, D)$, contains all the patterns in the dataset, D, that are translationally equivalent to the pattern, P. Suppose T contains n translationally equivalent occurrences of the pattern, P, and that P contains m points. There



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

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

Fig. 3. The SIA algorithm. (a) A small dataset that could be provided as input to the SIA or SIATEC algorithms. (b) The vector table computed by SIA for the dataset in (a). Each entry in the table gives the vector from a point to a lexicographically later point. Each entry has a pointer back to the origin point used to compute the vector. (c) The list of $\langle\text{vector}, \text{point}\rangle$ pairs that results when the entries in the vector table in (b) are sorted into lexicographical order. If this list is segmented at points at which the vector changes (as indicated by the boxes surrounding entries in the column headed "Datapoint"), then the set of points in the entries within a segment form the MTP for the vector for that segment. This means that all the MTPs can be obtained simply by scanning this list once (i.e., in $O(n^2)$ time, since the list has length $n(n-1)/2$).

are at least two ways in which one can specify T. First, one can explicitly specify each of the n patterns in T by listing all of the m points in each pattern. This requires one to write down mn, k-dimensional points or kmn numbers. For example, using this encoding method, the TEC in the left-hand graph in Fig. 4 would be specified as

$$\{\{\langle 1,1\rangle, \langle 2,2\rangle, \langle 4,2\rangle\}, \{\langle 2,6\rangle, \langle 3,7\rangle, \langle 5,7\rangle\}, \{\langle 3,5\rangle, \langle 4,6\rangle, \langle 6,6\rangle\}\} .$$

Alternatively, one can explicitly list the m points in just one of the patterns in T (e.g., P) and then give the $n-1$ vectors required to translate this pattern onto the other patterns in T. This requires one to write down m, k-dimensional points and $n-1$, k-dimensional vectors—that is, $k(m+n-1)$ integers. If n and m are both greater than one, then $k(m+n-1)$ is less than kmn, implying that the second method of specifying a TEC gives us a *compressed* encoding of the TEC. Thus, if a dataset contains at least two occurrences of a pattern containing at least two points, it will 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 pattern in T, and V is the set of vectors that translate P onto the other patterns in T. When a TEC, $T = \langle P, V\rangle$, is represented in this way, we call V the *set of translators* for the TEC and P the

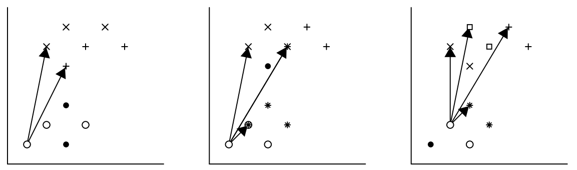


Fig. 4. Examples of translational equivalence classes (TECs). In each graph, the pattern of circles is translatable by the vectors indicated by the arrows. The TEC of each pattern of circles is the set of patterns containing the circle pattern itself along with the other patterns generated by translating the circle pattern by the vectors indicated. The covered set of each TEC is the set of points denoted by icons other than filled black dots.

TEC's *pattern*. We also denote and define the *compression ratio* of a TEC, $T = \langle P, V\rangle$ as follows:

$$\text{CR}(T) = \frac{|\text{COV}(T)|}{|P| + |V| - 1} \,. \tag{1}$$

In this paper, the pattern, P, of a TEC used to encode it as a $\langle P, V\rangle$ pair will be assumed to be the lexicographically earliest occurring member of the TEC (i.e., the one that contains the lexicographically least point). As an example, the TEC in the left-hand graph in Fig. 4, would be encoded as

$$\langle\{\langle 1,1\rangle, \langle 2,2\rangle, \langle 4,2\rangle\}, \{\langle 1,5\rangle, \langle 2,4\rangle\}\rangle \,,$$

giving a compression ratio of $9/5$ for this TEC. Meredith *et al.* [2,3] describe an algorithm, called SIATEC ("SIA+TECs"), 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 translationally equivalent occurrences of each MTP). For a k-dimensional dataset containing n points, SIATEC has a worst-case running time of $O(kn^3)$ and uses $O(kn^2)$ space.

2.3. Compactness

Meredith *et al.* [2, p. 340] observe that, although many important musical patterns relate to MTPs, many MTPs do *not* obviously relate to important musical patterns. Moreover, a typical short piece of classical music containing a few hundred notes may contain tens of thousands of MTPs, whereas an analyst would typically aim to explain such a piece in terms of only a few tens of patterns at most. Meredith *et al.* [2,4] suggest a number of strategies for isolating what they call "theme-like or motif-like" patterns in a piece of music. In particular, they suggest that the extent to which a pattern will be perceived to be "theme-like" may depend on its *compactness* defined 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" [4, p. 8]. Meredith *et al.* [4] acknowledge that 'the region spanned by a pattern' can be defined in a number of different ways, including, for example, the time segment spanned by the pattern or the pattern's bounding box or convex hull in the pitch-time representation.

3. RELATED WORK

The COSIATEC algorithm was first sketched by Meredith *et al.* [4]. They were also the first to suggest using the degree of compression achievable by expressing a TEC as a ⟨pattern,vector set⟩ pair as a heuristic for selecting theme-like patterns. They obtained encouraging results when they used their implementation of COSIATEC to analyse J. S. Bach's 15 *Two-Part Inventions* (BWV 772–786), noting, in particular, that the subjects of the inventions tended to be discovered on the early iterations of the algorithm.

Forth [5, 6] presented an algorithm, inspired by COSIATEC, that resembles the SIATECCompress algorithm to be described below. The first step in Forth's algorithm is to run SIATEC on the input dataset to generate a sequence of MTP TECs, $\mathbf{T} = \langle T_1, T_2, \ldots T_n\rangle$. The algorithm then post-processes the output of SIATEC to compute a cover for the input dataset. A weight, W_i, is assigned to each TEC, T_i, to produce a corresponding sequence of weights, $\mathbf{W} = \langle W_1, W_2, \ldots W_n\rangle$. W_i is intended to be a measure of the "structural salience" [6, p. 41] of the patterns in the TEC, T_i, and it is defined as $W_i = w'_{\text{cr},i} \cdot w'_{\text{compV},i}$ where $w'_{\text{cr},i}$ and $w'_{\text{compV},i}$ are normalized values representing the compression ratio and compactness of T_i. Having computed the sequence of weights, $\mathbf{W}$, Forth's algorithm then attempts to select a subset of the covered sets of the TECs in $\mathbf{T}$ that covers the input dataset and maximises the product of the coverage and weight of each TEC used in the encoding generated.

Collins *et al.* [7] claim that previous algorithms based on SIA are subject to what they call the 'problem of isolated membership'. This problem is defined to occur when "a musically important pattern is contained *within* an MTP, along with other temporally isolated members that may or may not be musically important" [7, p. 6]. Collins *et al.* claim that "the larger the dataset, the more likely it is that the problem will occur" and that it could prevent the SIA-based algorithms from "discovering some translational patterns that a music analyst considers noticeable or important". Collins *et al.* [7, p. 6] propose that this problem can be solved by taking each MTP computed by SIA (sorted into lexicographical order) and 'trawling' inside this MTP "from beginning to end, returning subsets that have a compactness greater than some threshold a and that contain at least b points." This method is implemented in an algorithm that they call SIACT, which first runs SIA on the dataset and then carries out 'compactness trawling' (hence "SIA*CT*") on each of the MTPs found by SIA.

Collins *et al.* [8] obtained empirical evidence for the hypothesis proposed in earlier work that compactness and compression ratio both play an important rôle in determining perceived importance or salience of a musical pattern. In Collins *et al.*'s study, twelve music theory undergraduates were asked to rate the importance and/or noticeability of 90 repeated patterns in Chopin's Mazurkas (each student rated 30 of the 90

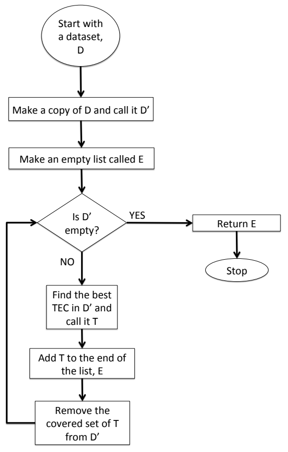

Start with a dataset, D
Make a copy of D and call it D′
Make an empty list called E
Is D′ empty?
YES
Return E
Stop
NO
Find the best TEC in D′ and call it T
Add T to the end of the list, E
Remove the covered set of T from D′

Fig. 5. The COSIATEC algorithm.

patterns). Twenty-nine quantifiable features were then calculated for each of the patterns. Some of these features were novel while others had been proposed in earlier work. Collins *et al.* found that they could explain over 70% of the variation in the ratings obtained in their experiment using just three features of the patterns, including pattern compactness and TEC compression ratio (as defined in Eq. 1 above).

In an attempt to improve on the precision and running time of SIA, Collins [9, pp. 282–283] defined a new, SIA-based algorithm called SIAR ("Structure induction algorithm for r superdiagonals"). Instead of computing the whole region below the leading diagonal in the vector table for a dataset (as in Fig. 3(b)), SIAR only computes the first r subdiagonals of this table. This is approximately equivalent to running SIA with a sliding window of size r [9, p. 168].

4. COSIATEC

COSIATEC (see Fig. 5) is a greedy, point-set compression algorithm, based on SIATEC [4]. COSIATEC takes a dataset, D, as input and computes a compressed encoding of D in the form of an ordered set of MTP TECs, E, such that $D = \bigcup_{T \in E} \mathrm{COV}(T)$ and, for all $T_1, T_2 \in E, T_1 \neq T_2$, $\mathrm{COV}(T_1) \cap \mathrm{COV}(T_2) = \emptyset$. That is, COSIATEC strictly partitions a dataset D into the covered sets of a set of MTP TECs, with each TEC represented as a $\langle$pattern, translator set$\rangle$ pair, as described in section 2.2 above.

As shown in Fig. 5, COSIATEC begins by making a copy of the input dataset which it stores in the variable D'. It also initializes E to an empty list. It then carries out a number of iterations of a loop. On each iteration, the algorithm finds the "best" MTP TEC in D', stores this in T and adds T to the end of E. It then removes the set of points covered by T from D'. When D' is empty, the algorithm terminates, returning the list of MTP TECs, E.

Finding the best TEC on each iteration involves using SIATEC to compute the MTP TECs in D' and selecting the TEC that has the best compression ratio. If more than one TEC has the maximum compression ratio, then the one with the highest compactness is chosen. If two or more TECs tie on both compression ratio and compactness, then the TEC with the highest coverage is chosen.

5. SIATECCompress

COSIATEC runs SIATEC on each iteration of its 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. On the other hand, because COSIATEC strictly partitions the dataset into non-overlapping MTP TEC covered sets, it tends to achieve relatively high compression ratios for many point-set representations of musical pieces (typically from 2–4 for a Bach keyboard fugue, for example).

Like COSIATEC, SIATECCompress is a greedy compression algorithm based on SIATEC that computes an encoding of a dataset in the form of a union of TEC covered sets. However, like Forth's algorithm, SIATECCompress runs SIATEC only *once* to get a list of TECs in decreasing order of quality. It then works its way down this list, selecting TECs to include in the encoding, until the input dataset is covered. The points covered by each selected TEC are not removed from the dataset after selection, thus SIATECCompress does not typically 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 in practice on much larger datasets. As in COSIATEC, TEC A is considered better than TEC B if A has a higher compression ratio than B, or if A and B have the same compression ratio but A has more compact patterns, or if A and B have the same compression ratio and compactness but A covers more points than B.

6. EVALUATION

SIATECCompress, COSIATEC, Forth's algorithm and variants of these algorithms incorporating Collins *et al.*'s compactness trawler and SIAR were compared on a classification task. Each algorithm was used as a compressor to classify a corpus of 360 Dutch folksong melodies from the Nederlandse Liederenbank[1] into "tune families" [10] using normalized compression distance [1, p. 664], a 1-nearest-neighbour classifier and leave-one-out cross-validation. The results are shown in Table 1. Java implementations of the algorithms and the code used to generated the results are available online.[2]

[1]http://www.liederenbank.nl
[2]http://chromamorph.googlecode.com

Algorithm	*SR*	*CR on AC*	*CR on file pairs*
COSIATEC	0.8389	1.5791	1.6670
COSIARTEC	0.8361	1.5726	1.6569
COSIARCTTEC	0.7917	1.4547	1.5135
COSIACTTEC	0.7694	1.4556	1.5138
Forth	0.6111	1.2645	1.2667
SIARCTTECCompress	0.5750	1.3213	1.3389
SIATECCompress	0.5694	1.3360	1.3256
SIACTTECCompress	0.5250	1.3197	1.3381
SIARTECCompress	0.5222	1.3283	1.3216
BZIP2	0.1250	2.7678	3.5061

Table 1. Results of using compression algorithms to classify Dutch folksongs into families using NCD, 1-NN and leave-one-out cross-validation. Algorithms with names containing "SIAR" employed the SIAR algorithm in place of SIA. Algorithms with names containing "CT" used Collins *et al.*'s [7] compactness trawler. The column headed SR gives the classification success rate—i.e., the proportion of songs in the corpus correctly classified. The third and fourth columns give the mean compression ratio achieved by the algorithm over, respectively, the corpus and the file-pairs used to compute the compression distances.

7. CONCLUSIONS

Table 1 suggests that algorithms based on COSIATEC performed markedly better on this song classification task than those based on SIATECCOMPRESS or Forth's algorithm. Using SIAR instead of SIA and/or incorporating compactness trawling reduced the performance of COSIATEC, however, using both together, slightly improved the performance of SIATECCOMPRESS. Forth's algorithm performed slightly better than SIATECCOMPRESS, suggesting that it may be worth incorporating Forth's covering strategy into COSIATEC. The results obtained using one of the best general-purpose compressors (BZIP2) were much poorer than those obtained using the SIA-based algorithms, suggesting that general-purpose compressors fail to capture certain perceptually and analytically important musical structure. The SIA-based algorithms that achieved better compression ratios tended to perform better on this task. However, the best compressor (BZIP2) produced the worst classifier. Compression ratio alone was thus not a good indicator of classification accuracy.

Acknowledgements

Peter van Kranenburg kindly supplied the data files for the annotated corpus of the Nederlandse Liederenbank (`http://www.liederenbank.nl`). The work was carried out within the "Learning to Create" project (Lrn2Cre8) which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 610859.

8. REFERENCES

[1] Ming Li and Paul Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer, Berlin, third edition, 2008.

[2] David Meredith, Kjell Lemström, and Geraint A. Wiggins, "Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music," *Journal of New Music Research*, vol. 31, no. 4, pp. 321–345, 2002.

[3] David Meredith, Geraint A. Wiggins, and Kjell Lemström, "Pattern induction and matching in polyphonic music and other multi-dimensional datasets," in *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI2001)*, N. Callaos, X. Zong, C. Verges, and J. R. Pelaez, Eds., 2001, vol. X, pp. 61–66.

[4] 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.

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

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

[7] Tom Collins, Jeremy Thurlow, Robin Laney, Alistair Willis, and Paul H. Garthwaite, "A comparative evaluation of algorithms for discovering translational patterns in baroque keyboard works," in *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010), Utrecht, The Netherlands, 9–13 August 2010*, 2010, pp. 3–8.

[8] Tom Collins, Robin Laney, Alistair Willis, and Paul H. Garthwaite, "Modeling pattern importance in Chopin's Mazurkas," *Music Perception*, vol. 28, no. 4, pp. 387–414, 2011.

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

[10] Peter van Kranenburg, Anja Volk, and Frans Wiering, "A comparison between global and local features for computational classification of folk song melodies," *Journal of New Music Research*, vol. 42, no. 1, pp. 1–18, 2013.