Using General-Purpose Compression Algorithms for Music Analysis

Corentin Louboutin
corentin.louboutin@ens-rennes.fr
École Normale Supérieure de Rennes, France

David Meredith
dave@create.aau.dk
Aalborg University, Denmark

Abstract

General-purpose compression algorithms encode files as dictionaries of substrings with the positions of these strings' occurrences. We hypothesized that such algorithms could be used for pattern discovery in music. We compared LZ77, LZ78, Burrows-Wheeler and COSIATEC on classifying folk song melodies. A novel method was used, combining multiple viewpoints, the *k*-nearest-neighbour algorithm and a novel distance metric, *corpus compression distance*. Using single viewpoints, COSIATEC outperformed the general-purpose compressors, with a success rate of 85%. However, by combining 8 of the 10 best-performing viewpoints, including seven that used LZ77, the success rate rose to over 94%. In a second experiment, we compared LZ77 with COSIATEC on the task of discovering subject and countersubject entries in fugues by J. S. Bach. When voice information was absent in the input data, COSIATEC outperformed LZ77 with a mean F1 score of 0.123, compared with 0.053 for LZ77. However, when the music was processed a voice at a time, the F1 score for LZ77 more than doubled to 0.124. We also discovered a significant correlation between compression ratio and F1 score for all the algorithms, supporting the hypothesis that the best analyses are those represented by the shortest descriptions.

Corresponding author David Meredith, AD:MT, Aalborg University, Rendsburggade 14, 9000 Aalborg, Denmark. Tel: +45 99408092. Fax: +45 99402671. email: dave@create.aau.dk.

1 Introduction

In this paper, we explore the possibility of using general-purpose text compression algorithms for analysing symbolic music data. Meredith (2012, 2013a,b, 2014a), inspired by the theory of Kolmogorov complexity (Kolmogorov, 1965; Li and Vitányi, 2008), hypothesized that the simplest and shortest descriptions of any musical object are those that describe the best possible explanations for the structure of that object. Meredith et al. (2003) developed a geometric pattern discovery and compression algorithm, COSIATEC, which finds maximal repeated patterns in a point-set representation of a piece of music and uses these to produce a losslessly compressed encoding of the input piece. Meredith (2014a) evaluated COSIATEC by using it to classify the melodies in the *Annotated Corpus* of Dutch folk songs (Nederlandse Liederenbank, NLB) (Grijp, 2008b; van Kranenburg et al., 2013) using *normalized compression distance* (NCD), the *1-nearest-neighbour* classification algorithm and *leave-one-out* cross-validation (i.e., each melody of the corpus is associated with the class of the most similar melody in the rest of the corpus). Using this general method, with no specific tuning of parameters for this particular task, he achieved a classification success rate of 84%.

The success of COSIATEC seemed to support the hypothesis that compressed encodings of melodies capture perceptually important structure in them. Like COSIATEC, general-purpose compression algorithms, such as *Lempel-Ziv-77* (LZ77) (Ziv and Lempel, 1977), are also based on the redundancy of an input sequence. This suggests that they might be useful for finding musically relevant patterns. In the study reported in this paper, we therefore explored this possibility by adapting general-purpose text compression algorithms for use with symbolic music representations, and then using the compressed encodings they produce for music classification. We also investigated the effect of using different representation schemes or *viewpoints* (Conklin and Witten, 1995) on the efficiency and effectiveness of these algorithms.

In a study by van Kranenburg et al. (2013), a classification method based on local features (Conklin, 2013a,b; Hillewaere et al., 2009; van Kranenburg et al., 2013), such as pattern similarity, outperformed methods that depended primarily on global features (Freeman and Merriam, 1956; Hillewaere et al., 2009; van Kranenburg et al., 2013), such as tonality, first and last note of a melody, average pitch and so on. Moreover, Conklin (2013a,b) recently showed that combining both local and global features using the *multiple viewpoint* approach yielded better results in a classification task than using just a single feature or viewpoint. This approach has also produced good results on prediction and generation of music (Conklin and Witten, 1995; Pachet, 2003).

In the present paper, we describe and analyse derivative versions of four compression algorithms: *Burrows-Wheeler* (Burrows and Wheeler, 1994), *Lempel-Ziv-77* (Ziv and Lempel, 1977), *Lempel-Ziv-78* (Ziv and Lempel, 1978) and COSIATEC. The first three of these algorithms are general-purpose compression algorithms that were developed for text compression. The fourth, COSIATEC, was specifically developed for analysing musical objects represented as point sets, but can, in fact, be applied in general to multi-dimensional point-set data. We use these four algorithms to compress sequences of two-dimensional points, treated as one-dimensional sequences of symbols from the alphabet $\mathbb{Z}^2$. For this reason, the examples presented below will use letters as symbol labels instead of actual two-dimensional points. The goal is to preserve the design of the text compression algorithms, but present the musical data in a way that allows these algorithms to find important repeated patterns.

Sections 2 to 5 present the four algorithms considered in our study. We then present a new classification method that combines the *multiple viewpoints* approach (Conklin, 2013b) and the *k-nearest-neighbour* algorithm. The last section compares all the algorithms considered, each combined with various representations. This evaluation is done on two tasks:

- a classification task run on the Dutch Song Database, *Onder der Groene linde* (Grijp, 2008a), with the new classification method; and
- a pattern discovery task for LZ77 and COSIATEC on the 24 fugues from the first book of J. S. Bach's *Das Wohltemperirte Clavier*.

2 Burrows-Wheeler

One of the most widely-used, general-purpose compression algorithms is *bzip2* (Seward, 2000), which is based on the work of Burrows and Wheeler (1994) (see also Sayood, 2012). The Burrows-Wheeler algorithm uses a transformation on the input sequence along with entropy coding. The Burrows-Wheeler algorithm outperforms the standard GNU compression program, *gzip* (`http://www.gzip.org`), in terms of both speed and compression. We therefore decided to explore the possibility of adapting it for pattern discovery in note sequences.

The algorithm consists of three parts:

1. The *Burrows-Wheeler transform.* This step executes a permutation of the input sequence that improves the compression effect of the following step.

row						T
0	a	b	a	n	a	n
1	a	n	a	b	a	n
2	a	n	a	n	a	b
3	b	a	n	a	n	a
4	n	a	b	a	n	a
5	n	a	n	a	b	a

Figure 1: Example of a matrix used by the Burrows-Wheeler transform.

2. *Move-to-front coding.* This is a transformation that can improve the performance of entropy coding such as Huffman coding. It also has a high compression effect.

3. *Huffman* or *arithmetic coding.*

We implemented the first two parts of the algorithm, but omitted the final coding step as it was clear that this would improve neither the success rate nor the compression ratio for the *Annotated Corpus*. This is because we use a string representation and there are only a few notes in a melody, which implies that a radix-10 representation, that uses fewer characters, would work better than a radix-2 representation.

2.1 Burrows-Wheeler Transform

The *Burrows-Wheeler transform* performs a permutation on the input string. The aim of this permutation is to bring equal elements closer together. This permutation increases the probability of finding a character c at a point in a sequence if c already occurs near this point. This can often result in better compression.

The Burrows-Wheeler transform uses an $n \times n$ matrix where n is the length of the input string S (see Figure 1). The elements of this matrix are points in S. Each row is a distinct cyclic shift of S. There is therefore at least one row that is equal to the input. The rows are then sorted into lexicographic order. The output of the algorithm is a pair (T, i), where T is the last column of the matrix and i is the index of a row corresponding to S (usually, there is only one such row).

An example of such a sorted matrix using the input string $S = banana$ is shown in Figure 1. As S appears in row 3, the output is then the pair formed by the string of the last column and this index: $(nnbaaa, 3)$. In this example, characters that are equal are regrouped together. However, this is not always the case, as can be seen in Burrows and Wheeler's (1994) own example, *abraca*, which is transformed into *caraab*.

2.2 Move-to-front Coding

The second step in the algorithm is to encode the string returned by the Burrows-Wheeler transform using move-to-front coding. This step takes a string, T, as input and returns a vector, R, of integers. This algorithm needs to know the alphabet, Y, of the input, so the first step consists of an iterative algorithm that builds the alphabet by reading the input string from left to right, adding new characters to an initially empty alphabet.

The algorithm then builds R by executing Algorithm 1 (see below). It replaces each character, $T[i]$, by its index in the alphabet, Y, and then places that character at the beginning of Y. Applied to the string, *nnbaaa*, it first computes the alphabet, $Y = [n, b, a]$, and then returns the integer vector, $R = [0, 0, 1, 2, 0, 0]$.

```
Data: A sequence of letters $T$
Result: A vector of integers $R$
$Y$ = alphabet of $T$;
construct an empty array $R$ of length $|T|$;
for $i = 0$ to $|T| - 1$ do
    $R(i)$ = index of $T(i)$ in $Y$;
    Move $T(i)$ to the front of $Y$;
end
```

Algorithm 1: Move-to-front coding.

The input of this algorithm is such that, when a character appears, the probability that it has already appeared or will appear again is high. Therefore, the integer found by the first instructions of the loop will be lower than without the transform.

To ensure reversibility, the algorithm needs to return the alphabet, Y, as well as the integer vector, R, returned by the move-to-front coding algorithm and the index, i, returned by the Burrows-Wheeler transform.

3 Lempel-Ziv-77 (LZ77)

In 1977, A. Lempel and J. Ziv introduced a lossless, dictionary-based data compression algorithm, commonly called LZ77 (Ziv and Lempel, 1977). There have been some improvements proposed for this algorithm, such as LZMA which is used by the 7zip compressor (I. Pavlov, ND). However, some compressors such as ZPAQ, which is one of the best general-purpose compressors currently available (Mahoney, 2009), still continue to use the basic version of LZ77. LZ77 achieves compression by discovering repeated patterns in strings, coding repeated substrings by references to their occurrences (Sayood, 2012). This motivated us to explore its potential for discovering musically relevant patterns in note sequences.

The LZ77 algorithm uses a *sliding window* which consists of two parts: the *dictionary* part and the *look-ahead buffer*. The dictionary contains an already-encoded part of the sequence, and the look-ahead buffer contains the next portion of the input to encode. The size of each part is determined by two parameters: n, the size of the sliding window; and L_s, the maximal matching length (i.e., the size of the look-ahead buffer).

Before looking in detail at the working of LZ77, we first introduce some notation relating to strings. Let S_1 and S_2 be two strings. $S_1(i)$ denotes the $(i+1)$th element in S_1 (i.e., zero-based indexing is used). $S_1(i,j)$ is the substring from $S_1(i)$ to $S_1(j)$. S_1S_2 is the string obtained by concatenating S_1 and S_2. Finally, S_1^n denotes S_1 concatenated n times.

The main principle of LZ77 is to find the longest prefix of the look-ahead buffer that also has an occurrence which begins in the dictionary. The output is then a sequence of triples, $(p_i, l_i - 1, c)$, where p_i is a pointer to the first letter of the dictionary occurrence, $l_i - 1$ is the length of the prefix and c is the first character that follows the prefix in the look-ahead buffer.

LZ77 is an iterative algorithm. First it initializes a window, W, by filling the dictionary with a null letter (a in the examples below, however, in practice, we use the point $(0,0)$). The look-ahead buffer is then filled with the first L_s notes of the input sequence, S, to be encoded—that is,

$$W = a^{n-L_s} S(0, L_s - 1) \; .$$

The followings steps are then repeated until the whole sequence, S, is encoded:

1. Find $S_i = W(n - L_s, n - L_s + l_i - 2)$, the longest prefix of length $l_i - 1$ of the look-ahead buffer that also has an occurrence which begins at index p_i in the dictionary. When there is no prefix (i.e., $l_i = 1$), $p_i = 0$, and when there are several possible p_i, the smaller is taken. The dictionary occurrence of the prefix may run into the look-ahead buffer (and therefore overlap the prefix) if $l_i + p_i > n - L_s$.

2. Add the triple, $(p_i, l_i - 1, c)$, to the output string (radix-10 representation is used for p_i and l_i). c is the first character that follows the prefix in the look-ahead buffer—that is, $c = W(n - L_s + l_i - 1)$.

3. Shift the window and fill the end of the look-ahead buffer with the next l_i letters of the input sequence: W becomes $W(l_i, n)S(h_i + 1, h_i + l_i)$, where h_i is the index into S of the last element of W before the shift operation.

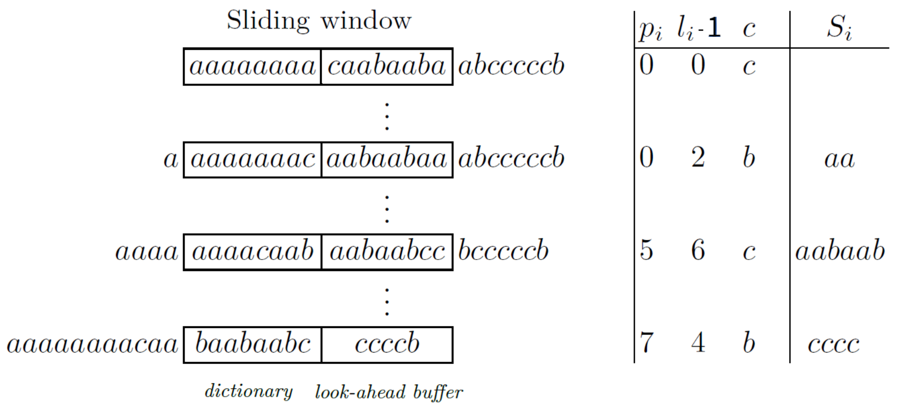

Figure 2: Sliding window used by the LZ77 algorithm.

Figure 2 shows LZ77 being used to encode the sequence *caabaabaabcccccb*. It first fills the dictionary with 'a' and the look-ahead buffer with the 8 first characters of the input sequence. Then there is no substring in the dictionary that begins with a c, so $l_i = 1$ and $p_i = 0$, the character following the prefix is c. Then, we shift the window by one (value of l_i) and obtain the state given in the second line. Here we find the prefix aa followed by b, so $l_i = 3$ and, as p_i can be any integer between 0 and 5, the algorithm returns the lowest one: $p_i = 0$. The window is then shifted by 3 and the state obtained is shown on line 3. Here, an overlapping occurs in which the prefix found, $aabaab$, begins in the dictionary and ends in the look-ahead buffer. On this step, the algorithm returns $(5, 6, c)$. The algorithm ends by doing one more step. Finally, the output is:

$$(0, 0, c)(0, 2, b)(5, 6, c)(7, 4, b) \ .$$

4 Lempel-Ziv-78

The Lempel-Ziv-78 (LZ78) algorithm is also a dictionary-based compression algorithm (Ziv and Lempel, 1978) (see also Sayood, 2012). However, in LZ78, the size of the dictionary is limited only by the amount of memory available. Many later compression algorithms have been based on LZ78, perhaps most notably the Lempel-Ziv-Welch (LZW) algorithm (Welch, 1984), which is used by the basic Linux command *compress*. However, as LZW needs to store the input alphabet with the dictionary, and as the input alphabet in our case is $\mathbb{Z}^2$, we preferred to use the basic version of LZ78.

The principle of LZ78 is to fill an explicit dictionary with substrings of the input. A feature of this algorithm is that the dictionary is the same at encoding and decoding.

	Dictionary	
Output	Index	Entry
(x, c)	0	c
(x, a)	1	a
$(1, b)$	2	ab
$(1, a)$	3	aa
(x, b)	4	b
$(3, b)$	5	aab
$(0, c)$	6	cc
$(6, c)$	7	ccc
$(4, \epsilon)$	8	

Figure 3: Example of sequence encoding with the LZ78 algorithm.

LZ78 works in four steps:

1. Create an empty substring B and extend it by adding characters of the input S until B does not appear in the dictionary.
2. Add the pair (i, c) to the output, where i is the last index met (i.e., the index corresponding to the longest match of B in the dictionary) and c is the last character added. In practice, when $i = -1$, the algorithm returns (x, c). This improves the compression ratio a little because it uses one character, whereas "-1" uses two.
3. Add B to the dictionary.
4. Set B to the empty string and repeat the steps until the whole input is encoded.

Figure 3 illustrates the encoding of the sequence *caabaabaabcccccb* with LZ78. When the algorithm begins, the dictionary is empty, therefore the two first letters encountered (c and a) are directly added into it and the returned index is -1 (encoded as 'x'). Then a is added to an empty B, but as a is already in the dictionary, the algorithm adds also b, producing $B = ab$ which is not in the dictionary. The output is then $(1, b)$, the index of the longest match (a) in the dictionary and the last character of B. It also adds B to the dictionary as a new substring encountered. The details of the remainder of the encoding process are tabulated in Figure 3.

5 COSIATEC

Unlike the preceding algorithms, COSIATEC (Meredith et al., 2003; Meredith, 2006, 2013b, 2014a) has not, to date, been used for general-purpose compression. This algorithm takes as input a set of points, D, in any number of dimensions, called a *dataset*, and outputs a parsimonious encoding of this dataset in the form of a set of *translational equivalence classes* (TECs) of *maximal translatable patterns* (MTPs). Any set of points in a dataset, D, is called a *pattern*. A maximal translatable pattern in a dataset, D, for a given vector, $\mathbf{v}$, is the set of points in D that can be translated by $\mathbf{v}$ onto other points in D. That is,

$$\mathrm{MTP}(\mathbf{v}, D) = \{p | p \in D \wedge p + \mathbf{v} \in D\}$$

where $p + \mathbf{v}$ is the point obtained by the translation of the point p by the vector $\mathbf{v}$. $\mathrm{MTP}(\mathbf{v}, D)$ is the subset of all points of D that have an image in D when translated by $\mathbf{v}$.

The TEC of a pattern, P, in a dataset, D, is the set of patterns in D onto which P can be mapped by translation. Every TEC has a *covered set* which is the union of the patterns that it contains. Each TEC in the output of COSIATEC is encoded compactly as a pair, $(\mathit{pattern}, \mathit{translator\ set})$, where the translator set is the set of vectors that map the pattern onto its other occurrences in the dataset. The possibility of encoding a TEC compactly in this way is the key to the algorithm's ability to compute a compressed encoding of an input dataset.

The algorithm used to find MTPs, called SIA, is fully described by Meredith et al. (2002), and will therefore not be reviewed here.

The equivalence relation used to build TECs, denoted by $\equiv_T$, is defined between two patterns P_1 and P_2 of a dataset D:

$$P_1 \equiv_T P_2 \iff (\exists \mathbf{v} | P_2 = P_1 + \mathbf{v})$$

where $P_1 + \mathbf{v}$ defines the set obtain by translating all points of P_1 by the vector, $\mathbf{v}$. The TEC of the pattern, $P \subseteq D$, is the equivalence class of P:

$$\mathrm{TEC}(P, D) = \{Q | Q \equiv_T P \wedge Q \subseteq D\}\ .$$

COSIATEC first runs the SIATEC algorithm (Meredith et al., 2002) to find MTP TECs (i.e., translational equivalence classes of the maximal translatable patterns in the input dataset). Each TEC in the output of SIATEC is represented by a pair (*pattern*, *translator set*). The TEC in the output of SIATEC which gives the best compression is then selected and added to the output encoding. The covered set of this TEC is then removed from the dataset and the process of running SIATEC and selecting the TEC that gives the best compression is repeated on the remaining dataset points. The process is repeated until every point in the dataset is covered by a TEC in the output encoding. The output encoding generated by COSIATEC is therefore a list of MTP TECs whose covered sets exclusively and exhaustively partition the input dataset.

6 Combined Representations Classification Method

In this section, we present the method we used to evaluate the compression algorithms described above. This method is based on Conklin and Witten's (1995) notion that "no single music representation can be sufficient for music" and that combining several representations—that is, *multiple viewpoints*—can produce a better model. With this method, good results have been achieved in prediction, generation and classification (Chordia et al., 2010; Conklin, 2013a,b; Pachet, 2003; Pearce et al., 2005). Our new method combines this multiple viewpoints approach with the well-known k-nearest-neighbour algorithm.

6.1 Definitions

We define a *representation* of a melody to be a reversible function that takes a string of two-dimensional points and returns a string of two-dimensional points. It preserves the size of the string and the sequence of points—that is, a point is replaced in the sequence by its new representation. Each representation we used is described in Table 1. We also used composition of transformations, $\circ$, as the composition on functions.

The *onset* of a note is the time of the start of the note and the *pitch* is the MIDI value of the note. Unless otherwise stated, all representations are applied on a string of $(\mathit{onset}, \mathit{pitch})$ points sorted into lexicographic order.

Here, a *viewpoint* is a pair, (Z, R), where Z is a compression algorithm and R is a representation. It can be seen as a function that takes a melody in the *pitch-time* representation and returns a string of characters: $Z \circ R$.

To be able to use 1-nearest-neighbour, Meredith (2014a,b) used *normalized compression distance* (Li et al., 2004). This is defined on a viewpoint, Z, and two melodies, s and s', as follows:

$$\mathrm{NCD}(Z, s, s') \; = \; \frac{|Z(s + s')| - \min\left(|Z(s)|, |Z(s')|\right)}{\max\left(|Z(s)|, |Z(s')|\right)}$$

where $|x|$ is the length of encoding x. This distance has two problems. First, the values are not restricted to being in the interval $[0, 1]$. Second, for two different compression

Name	Description
basic	The basic *pitch-time* representation—i.e., a string of (*onset*, *pitch*) points
int	A string of (*onset*, *pitch interval*) points: $$int(p_0) = p_0$$ $$int(p_n) = (p_n.onset,\ p_n.pitch - p_{n-1}.pitch)$$
*int*0	A string of (*onset*, *pitch interval from first note*) points: $$int0(p_0) = p_0$$ $$int0(p_n) = (p_n.onset,\ p_n.pitch - p_0.pitch)$$
intb	A string of (*onset*, *pitch interval pointer*) points: $$intb(p_0) = p_0$$ $$intb(p_n) = \begin{cases} (p_n.onset,\ p_n.pitch - p_{n-1}.pitch) & \text{if it is the first time the interval occurs,} \\ (p_n.onset,\ n - j) & \text{otherwise.} \end{cases}$$ where j is the index of the most recent occurrence of the interval in the input.
ioi	Inter-onset interval. $$int(p_0) = p_0$$ $$ioi(p_n) = (p_n.onset - p_{n-1}.onset,\ p_n.pitch)$$
ioib	Same as intb but for inter-onset intervals.

Table 1: The viewpoints used in the experiments. p_i is the (i+1)th point in the basic representation.

algorithms on the same corpus, the distances will not be comparable. For example, in our evaluation, one of the algorithms gave values in the range $[0.5, 0.8]$, and another produced values in the range $[0.8, 1.2]$. We therefore define another distance, called *Corpus Compression Distance* (CCD), which depends on the corpus, C, used for classification. It has the feature that it computes values in the interval $[0, 1]$ for all algorithms. The CCD is defined by the following formula:

$$\mathrm{CCD}(s, s', Z, C) = \frac{\mathrm{NCD}(Z, s, s') - \min_{s_1,s_2 \in C \cup \{s\}} \mathrm{NCD}(Z, s_1, s_2)}{\max_{s_1,s_2 \in C \cup \{s\}} \mathrm{NCD}(Z, s_1, s_2) - \min_{s_1,s_2 \in C \cup \{s\}} \mathrm{NCD}(Z, s_1, s_2)} .$$

To evaluate the algorithms, we also examined the compression ratios achieved, since these appeared to be related to the classification success rate. The compression ratio, $CR(v, s)$, achieved by an algorithm that generates an encoding, v, for a melody, s, is defined by:

$$CR(v, s) = \frac{|s|}{|v|} .$$

Finally, the classification success rate is defined as follows:

$$SR = \frac{\text{number of correctly classified melodies}}{\text{number of melodies in the corpus}} .$$

6.2 Classification Method

The classification method takes a melody and a corpus as input and returns a class which aims to be the real tune family of the melody. For this, it computes a matrix, M, of the type developed by Conklin (2013a,b). The matrix is shown in Table 2. To fill this matrix, we use a function f that depends on:

- C, the known corpus (i.e., the labelled melodies);
- s, the melody to be classified (not yet labelled);
- j, the class (i.e., tune family) to evaluate;
- v, the viewpoint applied; and
- N, the number of nearest neighbours to consider.

This function, f, gives a measure of how similar the melody, s, is to its nearest neighbours that are in tune family j. The higher the value is, the higher the probability that s will

M	1	$\cdots$	j	$\cdots$	m
v_1					
$\vdots$			$\vdots$		
v_i		$\cdots$	$f(C,s,j,v_i,N)$	$\cdots$	
$\vdots$			$\vdots$		
v_n					
		$\cdots$	$g(j)$	$\cdots$	

Table 2: Table computed for the melody to be classified.

be in j. The value of f is given by the following formula:

$$f(C,s,j,v,N) = \sum_{s_i \in C_j^N(s)} \frac{1}{(\mathrm{CCD}(s,s_i,v,C)+\epsilon)^{Ni}}$$

where ϵ is a constant as low as we want, and:

$$C_j^N(s) \;=\; C_j \cap C^N(s)$$

where C_j is the subset of C which contains the melodies in class, j, and $C^N(s)$ is the N nearest neighbours of s in C. The primary purpose of the ϵ factor is to avoid divide-by-zero error, but the value and the placement of it under the power has little effect on the results. In practice, we use $\epsilon = 0.1$.

The bottom row in Table 2 gives the geometric mean, $g(j)$, of the values of f for the class j, weighted by the proportion of corpus melodies in class j, that is:

$$g(j) \;=\; \frac{|C_j|}{|C|}\sqrt[n]{\prod_{i=1}^{n} M_{i,j}}$$

where $|.|$ is used for the cardinality of sets. As this method is used with the leave-one-out strategy, s is neither in C nor C_j. Finally, we choose the class with the maximum value to classify s:

$$c^* \;=\; \operatorname*{argmax}_{c\in[1,m]} g(c)\ .$$

7 Results

The algorithms described above where first evaluated on the task of classifying melodies in the Dutch Song Database (`http://www.liederenbank.nl`). LZ77 and COSIATEC were then compared on the task of discovering subject and countersubject entries in the fugues in the first book of J. S. Bach's *Das Wohltemperirte Clavier*. The results of these experiments will now be presented and discussed.

7.1 Task 1: Classifying folk song melodies

In our first evaluation task, the algorithms described above were used to classify the melodies in the *Annotated Corpus* of Dutch folk songs, *Onder der groene linde* (Grijp, 2008a). This corpus is available on the website of the Dutch Song Database (`http://www.liederenbank.nl`) provided by the Meertens Institute. It consists of 360 melodies, each classified by expert musicologists into one of 26 tune families. Each family is represented by at least 8 melodies and not more than 27 melodies. Each melody is labelled in the database with the name of the family to which it belongs. Each of the melodies is monophonic and contains around 50 notes.

To classify each melody, we used the method described in Section 6 in combination with leave-one-out cross-validation. We tested the method first with single viewpoints separately and then with combined viewpoints. Appendix A describes how LZ77 parameters were chosen.

7.1.1 Single Viewpoint Classification

To evaluate our method, we first used it with single viewpoints. The method was used with $N = 8$. That is, the method only considered the first 8 nearest neighbours of the melody to classify. The reason for this value is that the smallest tune family has only 8 melodies and so a larger N would increase the error in the method. Leave-one-out cross-validation was then used to predict the tune family of each melody. The various representations used are described in Table 1.

As all melodies in this corpus are monophonic, if the basic representation were used (see Table 1) every symbol would be distinct, leading to no repeated substrings which would mean that general-purpose text compression algorithms would find no repeated patterns. These algorithms can therefore only work on representations that transform the onset values. As these algorithms need equality on points in order to compress the strings, if all onsets are distinct, this implies a compression ratio less than 1. Compression ratios less than 1 were associated with poor classification success rates, so all viewpoints with an average compression ratio less than 1 were discarded.

Conversely, COSIATEC cannot use representations that transform the onsets (*ioi*, *ioib* and combined—see Table 1). Those representations are good for LZ77, LZ78 and

Viewpoint	*1-NN Leave-one-out SR*	CR_{AC}	CR_{pairs}
(COSIATEC, *basic*)	0.8528	1.5794	1.6670
(LZ77, $int \circ ioi$)	0.8222	1.4597	1.6735
(LZ77, $ioi \circ ioi$)	0.8222	1.2108	1.3547
(LZ77, *ioi*)	0.8194	1.3075	1.4915
(LZ77, $int0 \circ ioi$)	0.8139	1.3769	1.5690
(LZ77, *ioib*)	0.7944	1.1188	1.2629
(LZ77, $int0 \circ ioib$)	0.7861	1.1806	1.3306
(COSIATEC, *int*)	0.7556	1.5266	1.6226
(LZ77, $ioi \circ ioib$)	0.7472	1.0088	1.1127
(LZ77, $int \circ ioib$)	0.7444	1.2389	1.4062
(BW, *ioi*)	0.7333	1.9627	2.2768
(BW, $int0 \circ ioi$)	0.7194	2.0732	2.3853
(BW, $int0 \circ ioib$)	0.7111	1.4192	1.5436
(LZ78, *ioi*)	0.6361	1.7542	1.9292

Table 3: Results of the classification method with single viewpoints, sorted into descending order by success rate. *SR* denotes success rate; CR_{AC} denotes mean compression ratio on *Annotated Corpus*; and CR_{pairs} is for the average compression ratio on pair files used to compute the NCDs.

BW because they create redundancy, but COSIATEC needs a set of distinct points in order to work. In fact it is a condition on the reversibility of COSIATEC. Therefore, those viewpoints were also discarded.

Table 3 shows the results obtained by using the classification method on each viewpoint separately (i.e., in each case, the table corresponding to Table 2 contained only one row). Only those algorithm-viewpoint combinations are listed that resulted in a success rate higher than 70% (along with the highest-scoring combination for LZ78). We can see that in terms of success rate, COSIATEC outperforms all of the other compression algorithms with a value of 0.8528 with the basic (*onset*, *pitch*) representation. The viewpoint (COSIATEC, *int*) achieves poorer results than the viewpoint (COSIATEC, *basic*). This implies that the patterns found are not the same with each representation. Therefore, it is very important to find the representation that provides the best success rate for a given algorithm.

LZ77 also produced very good results and we can see that it is good for several representations. In fact, eight of the ten best viewpoints use LZ77. However, this algorithm does not compress well for most of the representations. Conversely, the Burrows-Wheeler algorithm achieved good compression but did not perform so well in terms of classification.

The bottom row of Table 3 gives the best result achieved using LZ78. The average

First viewpoints	*Leave-one-out SR*
2	0.8833
3	0.9139
4	0.9250
5	0.9083
6	0.9083
8	0.9194
10	0.9333
12	0.9139
14	0.9139
$10'$	0.9444

Table 4: Results for the classification method with the n best viewpoints.

compression ratio is similar to that achieved with Burrows-Wheeler, but the success rate is very low. The reason is that the melodies are very short (approximately 50 notes), whereas LZ78 needs many notes to match long patterns. We would expect LZ78 to perform better on longer pieces such as fugues or sonata-form movements, since the patterns it finds in such longer data would be likely to be longer and more relevant (i.e., there would be more long patterns).

7.1.2 Combined Viewpoints Classification

Having tested the algorithms with single viewpoints, we then carried out an evaluation in which the best viewpoints were combined. We chose to use the combined representations method only on viewpoints that gave good results when used alone. We then tested different combinations to determine which viewpoints improved the result. Table 4 shows the success rates obtained by the combined representations method using the n viewpoints that performed best individually. All results are better than those obtained using single viewpoints (cf. Table 3). But, it seems that some viewpoints have a detrimental effect on success rate (e.g., $(\text{LZ77}, \mathit{int}0 \circ \mathit{ioi})$, $(\text{COSIATEC}, \mathit{int})$). The last result in Table 4, denoted by $10'$, is obtained by combining eight of the ten best viewpoints, omitting the two viewpoints $(\text{LZ77}, \mathit{int}0 \circ \mathit{ioi})$ and $(\text{COSIATEC}, \mathit{int})$.

All the above results show that the representation used is an important factor in the classification success rate achieved. Indeed, the representation has a large effect on both the accuracy of the classification method and the compression ratio. On the other hand, the results also suggest that general-purpose compression algorithms can be used to find musically relevant patterns in a melody. The best success rate obtained with our new method is 0.944.

Conklin (2013a,b) ran his own method on the same corpus and achieved a success rate of 0.967 with the arithmetic fusion function and 0.958 with the geometric one. We speculate that the difference may be due to the fact that he was additionally using duration information to build viewpoints while we only use pitch and note onset information.

7.2 Task 2: Discovering subject and countersubject entries in fugues

As LZ77 is based on pattern matching, we decided to evaluate it on the task of discovering entries of subjects and countersubjects in the fugues from the first book of J. S. Bach's *Das Wohltemperirte Clavier* (BWV 846–869). We compared the output generated by the algorithms with the ground-truth analyses provided by Giraud et al. (2013). Again, we tested the algorithm with several different representations: $int \circ ioi$, $int \circ ioib$, ioi, $int0 \circ ioi$, $ioi \circ ioi$, $ioib$, $int0 \circ ioib$ and $ioi \circ ioib$. The parameters used for LZ77 were $n = 500$ and $L_S = 100$. The way these values were chosen is described in appendix A.

LZ77 was modified so that its output was in the form of a list of TECs, as this allowed for easier comparison with the ground-truth. This modification was effected in two steps, as follows:

1. Running LZ77 and returning a list of (*pattern*, *translator*) pairs instead of a list of $(p_i, l_i - 1, c)$ triples. Here, *pattern* is not the pattern in the viewpoint(s) that are being used for matching. It is actually the pattern of points in the basic pitch-time point-set representation that corresponds to the matched pattern in the viewpoint(s) being used in a particular case. *translator* is the translation vector between the first note of the occurrence of *pattern* in the equivalent dictionary and the first note of the equivalent look-ahead buffer in the pitch-time point-set representation.

2. Combining the pairs of the first step into sets of (*pattern*, *translator set*) pairs when the patterns are either equal or the patterns are longer than 3 notes and differ in either their first note or last note.

Figure 4 shows an example of an iteration of the first of these two steps. The match is done on the second representation which is $int \circ ioi$ but the algorithm returns the pair $((D, E, F), (2, 8))$.

Table 5 shows the results obtained with this modified LZ77 algorithm, compared with those obtained using COSIATEC. These results were obtained with the notes in each fugue sorted lexicographically (i.e., first by onset time, then by pitch height). For these

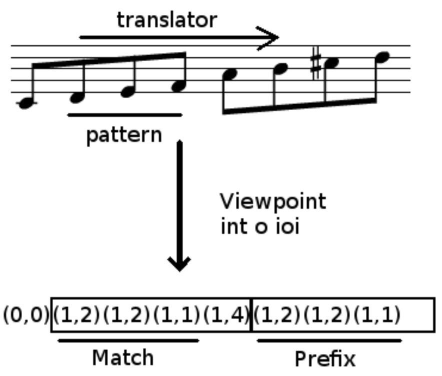

Figure 4: One iteration of LZ77, improved for pattern discovery.

Viewpoints	*TLF1*	*TLP*	*TLR*	*CR*
(COSIATEC, *basic*)	0.123	0.071	0.523	2.6404
(LZ77, $int \circ ioi$)	0.053	0.035	0.125	2.1976
(LZ77, $int \circ ioib$)	0.051	0.034	0.118	1.7464
(LZ77, *ioi*)	0.050	0.032	0.097	1.6876
(LZ77, $int0 \circ ioi$)	0.049	0.033	0.097	1.7305
(LZ77, $ioi \circ ioi$)	0.048	0.033	0.095	1.5397
(LZ77, *ioib*)	0.044	0.030	0.087	1.3929
(LZ77, $int0 \circ ioib$)	0.044	0.030	0.087	1.4255
(LZ77, $ioi \circ ioib$)	0.043	0.030	0.080	1.3374

Table 5: Results for the pattern discovery task on Bach's fugues with the different viewpoints based on LZ77 sorted by *TLF1* value. The notes were lexicographically sorted before applying the viewpoint, first by onset time then by pitch height with no regard for voice information. *TLP* is three-layer precision, *TLR* is three-layer recall, *TLF1* is three-layer F1 score and *CR* is average compression ratio.

results, the voice information for each note was ignored. The table shows the mean values over all the fugues for three-layer precision (TLP), three-layer recall (TLR) and three-layer F1 score (see Meredith, 2013c, for definitions of these measures). From this table, it can be seen that, when the notes in each fugue are sorted lexicographically with no regard for voice structure, COSIATEC outperforms all viewpoints based on LZ77, in terms of both compression ratio and similarity between the computed and ground-truth analyses.

This superior performance of COSIATEC can be explained by the fact that LZ77 only discovers maximal sub*strings*, whereas COSIATEC discovers maximal repeated point sets which correspond to maximal sub*sequences*. This makes COSIATEC's performance independent of the order in which the points are given in the input file and it is still possible for it to find themes consisting of contiguous notes within voices even if the points corresponding to these notes do not occur contiguously in the input file. On the

Viewpoints	*TLF1*	*TLP*	*TLR*	*CR*
$(\text{LZ77}, \mathit{int} \circ \mathit{ioi})$	0.124	0.073	0.521	3.7142
$(\text{COSIATEC}, \mathit{basic})$	0.123	0.071	0.523	2.6404
$(\text{LZ77}, \mathit{int} \circ \mathit{ioib})$	0.120	0.072	0.441	2.5008
$(\text{LZ77}, \mathit{ioi})$	0.114	0.073	0.298	1.9374
$(\text{LZ77}, \mathit{int}0 \circ \mathit{ioi})$	0.114	0.073	0.298	1.9553
$(\text{LZ77}, \mathit{ioi} \circ \mathit{ioi})$	0.108	0.068	0.280	1.7152
$(\text{LZ77}, \mathit{ioib})$	0.100	0.065	0.234	1.5428
$(\text{LZ77}, \mathit{int}0 \circ \mathit{ioib})$	0.100	0.065	0.234	1.5584
$(\text{LZ77}, \mathit{ioi} \circ \mathit{ioib})$	0.091	0.063	0.202	1.3424

Table 6: Results for the pattern discovery task on Bach's fugues with the different viewpoints based on LZ77 sorted by *TLF1* value. The notes were sorted first by voice, then by onset time and then by pitch height, so that adjacent notes within voices corresponded to adjacent symbols in the input data. *TLP*, *TLR*, *TLF1* and *CR* are defined as in Table 5.

other hand, LZ77 only discovers maximal sub*strings*, which implies that its performance depends crucially on the order in which the points are presented in the input file. It will thus only discover themes consisting of contiguous notes in a voice if the points corresponding to these notes occur contiguously in the input file.

To explore the effect that the sorting order of the input data has on COSIATEC and LZ77, we re-ran this experiment with the points sorted first by voice, then by onset time and then by pitch height. Under these sorting conditions, adjacent notes within a voice correspond to adjacent points in an input file, so we would expect LZ77's performance to increase but COSIATEC's performance to remain unchanged, since its performance is independent of the order in which the input data is sorted. Table 6 shows the results we obtained using this new sorting strategy. As can be seen in this table, changing the order in which the notes are encoded in the file, so that adjacent notes in the music correspond to adjacent symbols in the input file, typically *more than doubled* the mean F1 scores for LZ77. However, as predicted, changing the sorting order of the points in the input data had no effect on the performance of COSIATEC. In fact, when combined with the $\mathit{int} \circ \mathit{ioi}$ representation, under these input data sorting conditions, LZ77 outperformed COSIATEC substantially in terms of compression ratio and also marginally in terms of F1 score and precision. It can also be seen in Table 6 that changing the order in which the input file data is sorted does not affect the order of ranking of the different representations used with LZ77.

We can also see in Tables 5 and 6 that the compression ratio achieved seems to be related to pattern discovery performance. Indeed, there is a strong, highly significant, positive correlation, between compression ratio and TLF1 both for the data in Table 5 ($r = 0.8606, N = 9, p = 0.003$) and for the data in Table 6 ($r = 0.8542, N = 9, p = 0.003$)

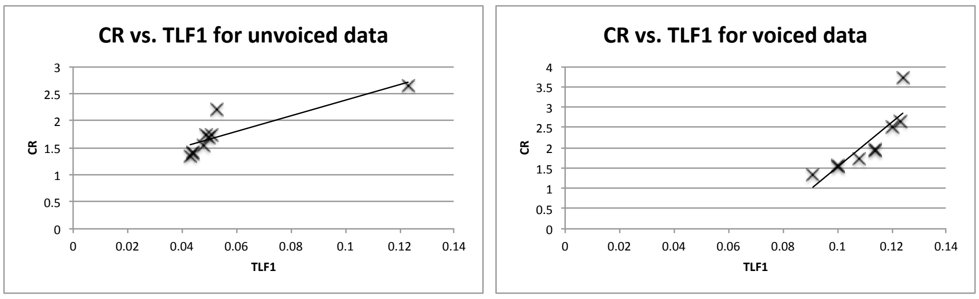

Figure 5: Graphs of compression ratio (CR) against three-layer F1 score (TLF1) for the values in Table 5 (left) and Table 6 (right). Linear trend lines are added as CR is highly positively correlated with TLF1 in both sets of data.

(see also the graphs in Figure 5). This supports the hypothesis, suggested by Kolmogorov complexity (Kolmogorov, 1965) and the minimum description length principle (Rissanen, 1978), that shorter descriptions represent better analyses.

8 Summary and conclusions

In this paper, we have presented a new classification method and applied it to the problem of classifying folk song melodies into tune families. The new method, based on *normalized compression distance*, the *k-nearest-neighbour* algorithm and the *multiple viewpoints* approach, was run with several compression algorithms and representations. The results showed that the specific way in which local features are represented has a large effect on classification performance. Of the algorithms tested, the geometric pattern discovery algorithm, COSIATEC, outperformed the general-purpose compression algorithms tested on the classification task when single viewpoints were used. However, LZ77 also performed very well with single viewpoints—indeed, eight of the ten best single-viewpoint-algorithm combinations tested used LZ77. Moreover, we were able to improve on the classification rate achieved by COSIATEC by combining eight of the ten best-performing combinations (7 of these used LZ77, the other used COSIATEC). This resulted in a classification success rate on the Annotated Corpus of Dutch folk song melodies of 94.4%. These results indicate that general-purpose compression algorithms (particularly LZ77) show great promise for use in melodic classification.

We also evaluated a modified version of LZ77 on the task of discovering subject and countersubject entries in fugues by J. S. Bach and, again, compared the results with those obtained using COSIATEC. COSIATEC out-performed LZ77 on this task when using the lexicographical order of notes without regard for voice structure, achieving a mean F1 score over the 24 fugues in the first book of *Das Wohltemperirte Clavier* of

0.123, compared with the best result of 0.053 for LZ77. However, we hypothesised that this disparity was due to the fact that, with the notes sorted into lexicographical order, LZ77 was rendered incapable of discovering repeated patterns consisting of sequences of contiguous notes within single voices (e.g., repeated thematic patterns such as subjects and countersubjects). To evaluate the effect on the algorithms of the order in which the notes are sorted in the input data, we re-ran the experiment with the notes in each fugue sorted first by voice, then by onset time and then by pitch height. This meant that contiguous sequences of notes in voices in the fugues now corresponded to contiguous sequences of symbols in the input data. Changing the order of notes in the input files in this way more than doubled the best F1 score for LZ77 to 0.124, marginally better than that of COSIATEC, which remained unchanged by the change in sorting order, as predicted. This result provides further evidence that LZ77 can successfully be used for analysing polyphonic music, provided that the data is represented appropriately and the algorithm is presented with the music one voice at a time. It should be noted, however, that COSIATEC performed as well as LZ77 on this pattern-discovery task, even when it was *not* provided with voice information.

Additionally, the results of our experiment on discovering fugal subject and countersubject entries revealed a highly significant, strong positive correlation between the compression ratio achieved by an algorithm and its pattern-discovery performance (measured in terms of three-layer F1 score) ($r = 0.86, N = 9, p = 0.003$). This result supports the hypothesis, suggested by the minimum description length principle and the concept of Kolmogorov complexity, that shorter descriptions of musical objects represent better analyses.

In conclusion, the results reported above suggest that general-purpose compression algorithms—and LZ77 in particular—could fruitfully be used for acquiring knowledge from representations of musical surfaces, that will allow for a variety of musicological tasks to be successfully automated.

Code

The source code of the Java implementations of the algorithms used to carry out the experiments reported in this paper are freely available at `chromamorph.googlecode.com`. For guidance on using these implementations, please contact the authors.

Acknowledgement

The work reported in this paper was carried out as part of the EU FP7 project, “Learning to Create” (Lrn2Cre8). The project Lrn2Cre8 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.

A Parameters of LZ77

The LZ77 algorithm needs two parameters to be set:

- n, the size of the sliding window;
- L_s, the size of the look-ahead buffer, $n - L_s$ is then the size of the dictionary.

The algorithm is based on the hypothesis that patterns will occur close together. Indeed there are a few cases where it does not work well. This can happen when the pattern that occurs again is no longer in the dictionary. The worst case of this is when the input is periodic with a period $p > n - L_S$ and contains a lot of different characters. As is shown in Figure 6, the algorithm cannot find the first letter in the dictionary and consequently cannot find any pattern. It is therefore important to find good parameters which correspond to the analyzed corpus.

$$\cdots abc \ \boxed{defghijk \,\big|\, abcdefgh} \ ijk\cdots$$

Figure 6: Worst case of LZ77 algorithm: periodic input of period p with $p > n - L_S$.

A.1 Parameters for the Annotated Corpus

To choose parameters for use on the folk-song classification task, we ran the classification method on the *Annotated Corpus* several times with the single viewpoint ($\mathrm{LZ77}_{n,L_s}, int \circ ioi$). Each pair of parameters (n, L_s) produced a success rate, an average compression ratio on melody files, and an average compression ratio on pair files. The results are reported in Table 7. This table shows that success rates tend to increase up to a limit with the dictionary size. The size of the look-ahead buffer seems to have a similar effect on compression ratios. Moreover, the last line shows poor compression ratio but a high classification success rate. Poor compression ratios are due to the fact that the size of the dictionary is more than 99, and so the returned pointers, p_i, can have three digits

n	L_S	*1-NN Leave-one-out SR*	CR_{AC}	CR_{pairs}
55	15	0.5472	1.3201	1.4433
70	20	0.7083	1.3376	1.4782
80	20	0.7444	1.3377	1.4875
100	10	$\boxed{0.8222}$	1.3151	1.4718
100	15	0.8056	1.3301	1.4881
100	25	0.7972	1.3378	1.4951
140	40	0.7611	1.4316	1.6710
150	30	0.8000	1.2680	1.4291

Table 7: Results of the classification method with the single viewpoint (LZ77$_{n,L_s}$, $int \circ ioi$). *SR* is Success Rate, CR_{AC} is the mean compression ratio on the *Annotated Corpus*, CR_{pairs} is the average compression ratio on pair files used to compute the CCDs.

instead of two. The high classification success rate can be explained by the fact that the dictionary is larger.

The best success rate is achieved with the parameters, $n = 100$ and $L_s = 10$. Taking these parameter values is a good choice because it increases the success rate and the size of the dictionary allows the algorithm to find similarities between the two melodies. Indeed, as the melodies have approximately 50 notes each, when LZ77 compresses the second melody, the first one is still in the dictionary. Of course, for another corpus, which contains larger pieces, it would be better to choose a larger value of n.

A.2 Parameters for discovering fugal subject and countersubject entries

As fugues are typically longer than folk song melodies, we would expect a small n to perform poorly. A fugue typically contains between 500 and 2000 notes. As the pattern discovery does not compare a fugue to another one, a dictionary size of 1000 is large enough. But considering that a pattern will appear more than one time in a piece, reducing the size of the dictionary will not decrease the compression ratio but will improve the execution time of the algorithm. Therefore, for the pattern discovery task on fugues, LZ77 was run with $n = 500$ and $L_S = 100$.

References

Burrows, M. and Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm. *SRC Research Report*, 124.

Chordia, P., Sastry, A., Mallikarjuna, T., and Albin, A. (2010). Multiple viewpoints modeling of tabla sequences. In *ISMIR*, volume 2010, page 11th.

Conklin, D. (2013a). Fusion functions for multiple viewpoints. In *Sixth International Workshop on Machine Learning and Music (MML)*, Prague, Czech Republic.

Conklin, D. (2013b). Multiple viewpoint systems for music classification. *Journal of New Music Research*, 42(1):19–26.

Conklin, D. and Witten, I. H. (1995). Multiple viewpoint systems for music prediction. *Journal of New Music Research*, 24(1):51–73.

Freeman, L. C. and Merriam, A. P. (1956). Statistical classification in anthropology: An application to ethnomusicology. *American Anthropologist*, 58(3):464–472.

Giraud, M., Groult, R., and Levé, F. (2013). Truth file for the analysis of Bach and Shostakovich fugues (2013/12/27 version). Available online at `http://www.algomus.fr/truth/fugues.truth.2013.12`.

Grijp, L. (2008a). Introduction. In L.P. Grijp and I. van Beersum, editors. *Under the Green Linden âĂŤ 163 Dutch Ballads from the Oral Tradition*, pages 18–27.

Grijp, L. P. (2008b). Introduction. In Grijp, L. P. and van Beersum, I., editors, *Under the Green Linden—163 Dutch Ballads from the Oral Tradition*, pages 18–27. Meertens Institute/Music & Words.

Hillewaere, R., Manderick, B., and Conklin, D. (2009). Global feature versus event models for folk song classification. In *ISMIR*, volume 2009, pages 729–733. ISMIR.

I. Pavlov (N.D.). LZMA description. `http://www.7-zip.org`. 2013.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition ofinformation'. *Problems of information transmission*, 1(1):1–7.

Li, M., Chen, X., Li, X., Ma, B., and Vitányi, P. M. (2004). The similarity metric. *Information Theory, IEEE Transactions on*, 50(12):3250–3264.

Li, M. and Vitányi, P. (2008). *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, Berlin, third edition.

Mahoney, M. (2009). Large text compression benchmark. *URL: http://www. mattmahoney. net/text/text. html*.

Meredith, D. (2006). 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. Available online at <http://drops.dagstuhl.de/opus/volltexte/2006/652>.

Meredith, D. (2012). Music analysis and Kolmogorov complexity. In *Proceedings of the 19th Colloquio di Informatica Musicale (XIX CIM)*, Trieste, Italy.

Meredith, D. (2013a). Analysis by compression: Automatic generation of compact geometric encodings of musical objects. In *The Music Encoding Conference (MEC 2013)*.

Meredith, D. (2013b). COSIATEC and SIATECCompress: Pattern discovery by geometric compression. In *International Society for Music Information Retrieval Conference*.

Meredith, D. (2013c). Three-layer precision, three-layer recall, and three-layer F1 score. `http://www.music-ir.org/mirex/wiki/2013:Discovery_of_Repeated_Themes_%26_Sections#Three-Layer_Precision.2C_Three-Layer_Recall.2C_and_Three-Layer_F1_Score`. 2013.

Meredith, D. (2014a). Compression-based geometric pattern discovery in music. In *Fourth International Workshop on Cognitive Information Processing (CIP 2014), 26–28 May 2014*, Copenhagen, Denmark.

Meredith, D. (2014b). Using point-set compression to classify folk songs. In *Fourth International Workshop on Folk Music Analysis (FMA 2014), 12–13 June 2014*, Bogazici University, Istanbul, Turkey.

Meredith, D., Lemström, K., and Wiggins, G. A. (2002). Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. *Journal of New Music Research*, 31(4):321–345.

Meredith, D., Lemström, K., and Wiggins, G. A. (2003). Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. In *Cambridge Music Processing Colloquium*.

Pachet, F. (2003). The continuator: Musical interaction with style. *Journal of New Music Research*, 32(3):333–341.

Pearce, M., Conklin, D., and Wiggins, G. (2005). Methods for combining statistical models of music. In *Computer Music Modeling and Retrieval*, pages 295–312. Springer.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–471.

Sayood, K. (2012). *Introduction to data compression.* Newnes.

Seward, J. (2000). The bzip2 and libbzip2 official home page. *Online]. http://www. bzip. org.*

van Kranenburg, P., Volk, A., and Wiering, F. (2013). A comparison between global and local features for computational classification of folk song melodies. *Journal of New Music Research*, 42(1):1–18.

Welch, T. A. (1984). A technique for high-performance data compression. *Computer*, 17(6):8–19.

Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530–536.