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 classification success rate of 85% on this task. However, by combining 8 of the 10 best-performing viewpoints, including seven that used LZ77, the classification 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 F_1 score of 0.123, compared with 0.053 for LZ77. However, when the music was processed a voice at a time, the F_1 score for LZ77 more than doubled to 0.124. We also discovered a significant correlation between compression factor and F_1 score for all the algorithms, supporting the hypothesis that the best analyses are those represented by the shortest descriptions.

Corresponding author David Meredith, 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 use of general-purpose text-compression algorithms for analysing symbolic music data. Drawing on the theory of Kolmogorov complexity (Kolmogorov, 1965; Li and Vitányi, 2008), it has been suggested previously 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, 2012, 2016). An *explanation* for the structure of an object is a description of the object that provides a hypothesis as to the process that gave rise to it. Typically, we want explanations to be as simple or short as possible, while also describing the explained object in as much detail as possible. This so-called "principle of parsimony" can be traced back to antiquity[1] and is known in common parlance as "Ockham's razor", after the mediaeval English philosopher, William of Ockham (ca. 1287–1347), who made several statements to this effect.

In more recent times, the parsimony principle has been formalized in various ways, including Rissanen's (1978) *minimum description length* (MDL) principle and Solomonoff's (1964a; 1964b) theory of inductive inference. The essential idea underpinning these techniques for learning from data is that *explanations* for data (i.e., ways of understanding it) can be derived from it in a bottom-up way, simply by compressing it. Indeed, Vitányi and Li (2000, p. 446) have shown that "data compression is almost always the best strategy" both for model selection and prediction. This provides motivation for the work presented in this paper, in which we explore the possibility that general-purpose compression algorithms can effectively be used to automatically derive successful explanations for (i.e., analyses of) the structures of pieces of music. More specifically, our work is based on the hypothesis that the shorter a description, the better it explains the object being described, suggesting the possibility of automatically deriving explanatory descriptions of objects (in our case, pieces of music) simply by compressing 'in extenso' descriptions of them. In the case of music, such an 'in extenso' description might be simply a list of the properties of the notes in a piece (e.g., the pitch, onset and duration of each note).

The minimum description length principle (MDL) as well as concepts related to MDL, such as relative entropy and mutual information (which originate in Shannon's (1948a; 1948b) information theory), have been used in several previous studies in the fields of computational music analysis and music information retrieval (e.g., Bimbot et al., 2012; Conklin and Witten, 1995; Mavromatis, 2005, 2009; Temperley, 2014; White, 2014). However, in these studies, stochastic models are typically assumed (e.g., HMMs (Mavromatis, 2005, 2009), Bayesian inference (Temperley, 2014), entropy-based models (Conklin and Witten,

[1]See, for example, chapter 25 of book 2 of Aristotle's *Posterior Analytics*.

1995)). That is, in these approaches, music is assumed to be the output of a random source that emits symbols in accordance with some (possibly context-dependent) probability distribution. In contrast, in this study we focus on non-probabilistic, dictionary-based compression algorithms, such as those based on the Lempel–Ziv algorithm (Ziv and Lempel, 1977, 1978) and bzip2 (Seward, 2010), that achieve compression by discovering repeated substrings in sequences and replacing occurrences of these substrings with low-information pointers to items in a dictionary. We focus on such dictionary-based algorithms rather than stochastic methods, because the former seem to relate more closely to analytical methods such as paradigmatic analysis (Ruwet, 1966; Nattiez, 1975), in which musical sequences are segmented and segments are compared and clustered into paradigms.

General-purpose text-compression algorithms have been used previously for computing the *normalized compression distances* (NCDs) (Li et al., 2004) between pairs of musical objects in classification and clustering tasks (Cilibrasi et al., 2004; Li and Sleep, 2004, 2005; Hillewaere et al., 2012). The results of these studies support the hypothesis that compressed encodings of melodies capture perceptually important structure in them. An assumption underlying most of these studies is that the specific compressor used should make little difference to the results. For example, Cilibrasi et al. (2004, p. 50) claim that their method is "robust under choice of different compressors". However, recent studies by Meredith (2014a,b, 2015, 2016) show that the choice of compressor used to measure NCD can have a large effect on performance in music classification tasks. For example, on the task of classifying the melodies in the *Annotated Corpus* of Dutch folk songs (Nederlandse Liederenbank, NLB) (Grijp, 2008; van Kranenburg et al., 2013), Meredith found that the classification success rate varied from 12.5% to 84%, depending on which compression algorithm was used to calculate the NCDs between the melodies. Moreover, these results did not seem to indicate a clear correlation between how well an algorithm compressed the melodies and how well it performed on classification. For example, the general-purpose text-compression algorithm, bzip2 (Seward, 2010), achieved an average compression factor of 2.76 but a success rate of only 12.5%; whereas the COSIATEC point-set compression algorithm (Meredith et al., 2003; Meredith, 2014b), which was originally designed for music analysis, achieved an average compression factor of only 1.58 but a classification success rate of 84%.

In this paper, we therefore investigate more closely the effect of choice of compressor on classification performance, by comparing four compression algorithms on two music-analytical tasks. The algorithms compared include three general-purpose, dictionary-based, text-compression algorithms and the COSIATEC point-set compression algorithm (which was originally designed for analysing music). We expect the general-purpose compressors to achieve better compression on average than COSIATEC, since they have been specifically designed to achieve good compression on many different types of data, whereas COSIATEC was designed to find patterns in music. Our motivating hypothesis (that shorter descriptions provide better explanations) leads us to expect a positive correlation between compression factor and classification accuracy, which, in turn, leads us to expect better classification success rates from the algorithms that achieve better compression. However, as mentioned above, this is not unambiguously supported by the results obtained by Meredith (2014a,b, 2015, 2016). We are therefore particularly interested in determining whether the general-purpose compressors, which typically achieve better compression factors than COSIATEC, are generally less successful than COSIATEC on music-analytical tasks, or if the poor classification success rate that Meredith achieved with bzip2 is atypical.

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 this paper, we therefore focus on local features and investigate the effect of using various different representation schemes (i.e., viewpoints), both separately and in combination, on the efficiency and effectiveness of the compression algorithms that are compared.

In section 2, we describe and analyse derivative versions of three general-purpose compression algorithms: *Burrows–Wheeler* (Burrows and Wheeler, 1994), *Lempel-Ziv-77* (Ziv and Lempel, 1977) and *Lempel-Ziv-78* (Ziv and Lempel, 1978). We also review the COSIATEC algorithm, which was specifically developed for analysing music represented as sets of points, but which could, 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 two-dimensional points. The goal was 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. In section 3, we then present a new classification method that combines the *multiple viewpoints* approach (Conklin, 2013b) and the *k-nearest-neighbour* algorithm. Finally, in section 4, we present the results obtained when the algorithms, combined with various input representations, were used to carry out two tasks:

1. a classification task run on the *Annotated Corpus* from the Dutch Song Database, *Onder der Groene linde* (Grijp, 2008), using the new classification method, described in section 3; and
2. 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 The algorithms

2.1 Burrows–Wheeler

One of the most widely-used, general-purpose compression algorithms is *bzip2* (Seward, 2010), 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 (at least as implemented in bzip2) typically achieves better compression than the standard GNU compression program, *gzip* (`http://www.gzip.org`).[2] 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.
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 all steps of the algorithm, but only used the first two parts, as the arithmetic (in our case, Huffman) coding step improved neither classification nor compression performance on the *Annotated Corpus.* We suspect this is due to the fact that the melodies we are analysing here are relatively short, which means that a radix-10 string representation, that uses fewer characters, performs better than a radix-2 representation (i.e., a bit-string). Nevertheless, by coding symbols in groups instead of individually, it is feasible that arithmetic coding might improve the results of the Burrows–Wheeler algorithm on the song classification task that we consider in this paper.

[2]See, for example, the results reported at `http://tukaani.org/lzma/benchmarks.html`.

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.

2.1.1 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*.

2.1.2 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. R is then built by executing the algorithm shown in Figure 2. 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, $R = [0, 0, 1, 2, 0, 0]$. The input of this algorithm is such that, when a

```
MOVE-TO-FRONT($T$)
1   $Y \leftarrow$ The alphabet of $T$
2   Construct an empty array $R$ of length $|T|$
3   for $i \leftarrow 0$ to $|T| - 1$
4       $R(i) \leftarrow$ The index of $T(i)$ in $Y$
5       Move $T(i)$ to the front of $Y$
6   return $R$
```

Figure 2: The move-to-front coding algorithm.

character appears, the probability that it has already appeared or will appear again is high. Therefore, the integer found in line 4 of Figure 2 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, and the index, i, returned by the Burrows–Wheeler transform.

2.2 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 (Pavlov, 2015). 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 and 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 a string consisting of n consecutive occurrences of S_1.

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 element of the dictionary occurrence, $l_i - 1$ is the length of the prefix and c is the first element 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 symbol (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 elements 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 element 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 elements 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 3 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 elements of the input sequence. Then there is no substring in the dictionary that begins with a c, so $l_i = 1$, $p_i = 0$ and the element 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:

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

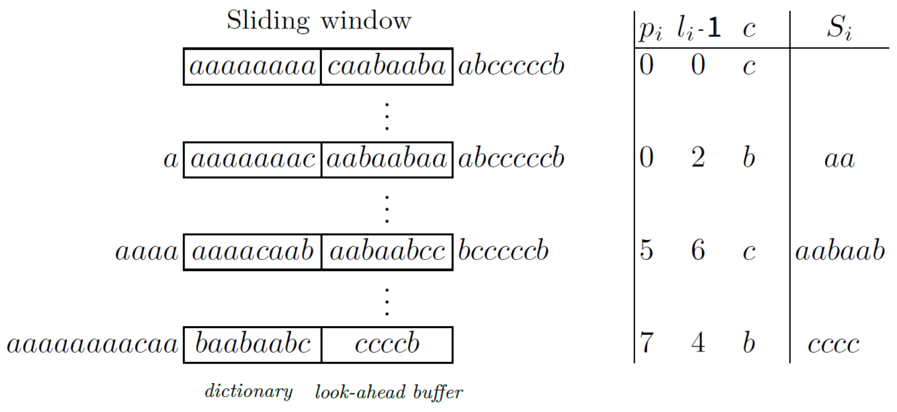

Figure 3: Sliding window used by the LZ77 algorithm.

2.3 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 in the dictionary, and as the input alphabet in our case is $\mathbb{Z}^2$ and therefore infinite, we preferred to use the basic version of LZ78.[3]

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.

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.[4]

3. Add B to the dictionary.

4. Set B to the empty string and repeat the steps until the whole input is encoded.

[3] Of course, in practice, our alphabet would be a finite subset of $\mathbb{Z}^2$, but this would still be very large and therefore significantly increase the size of the dictionary.

[4] In practice, when $i = -1$, the algorithm returns (x, c). This improves compression a little because it uses one character, whereas "-1" uses two.

Output	Dictionary	
	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 4: Example of sequence encoding with the LZ78 algorithm.

Figure 4 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 4.

2.4 COSIATEC

Unlike the preceding algorithms, COSIATEC (Meredith et al., 2003; Meredith, 2014b) 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,

$$\mathrm{MTP}(\mathbf{v}, D) = \{p \mid p \in D \land p + \mathbf{v} \in D\} , \tag{1}$$

where $p + \mathbf{v}$ is the point obtained by translating 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, (*pattern*, *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:

$$P_1 \equiv_T P_2 \iff (\exists \mathbf{v} | P_2 = P_1 + \mathbf{v}) \ , \tag{2}$$

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

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

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.

The COSIATEC algorithm was originally designed for analysing music, but it is actually a compression algorithm that can be applied to any data that can be represented as a set of points in a Euclidean space (of any dimensionality). For example, it could be used for text-compression by using a reversible mapping from A^* to $\mathbb{Z}^k$ where A is an alphabet. Such a mapping could, for example, consist of coding each symbol in a string, $S \in A^*$, as a 2-dimensional point, $\langle i, \ell \rangle$, where i is the index of the symbol's position in the string and ℓ is the index of the symbol in A.

3 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).

Meredith (2014b) compared the performance of several point-set compression algorithms on the task of classifying songs from the Dutch Song Database (Grijp, 2008) into tune families. For this classification, he used the 1-nearest-neighbour algorithm and normalized compression distance (NCD) (Li et al., 2004) and evaluated the classification success rate using leave-one-out cross-validation. As mentioned in the introduction, NCD has been used previously in several music classification studies (Cilibrasi et al., 2004; Hillewaere et al., 2012; Li and Sleep, 2004, 2005). Our new method combines the multiple viewpoints approach with the well-known k-nearest-neighbour algorithm using NCD to measure the similarity between melodies.

3.1 Representations

If $(\mathbb{Z}^2)^*$ is the set of strings of 2-dimensional points with integer co-ordinates, then we define a *representation* of a melody to be a function, $f : (\mathbb{Z}^2)^* \to (\mathbb{Z}^2)^*$, where f preserves the size of the string and the sequence of points—that is, a point is replaced in the sequence by its new representation. The function must be reversible if it is to be used for lossless compression, but for classification this is not necessary. Each representation we used is described in Table 1. We also used composition of transformations, $\circ$, as the composition on functions.

The viewpoint representations chosen for this study were based on those used by van Kranenburg et al. (2013) and Conklin (2013b). Van Kranenburg et al. (2013) discovered features that allow for the data from the folk song dataset to be classified with almost perfect accuracy. However, the musicologists who provided the "ground-truth" classification did not describe any explicit criteria or method that they used to determine the tune families to which they judged the songs to belong. Indeed, one of the principal motivations behind van Kranenburg et al.'s (2013) work was to discover the criteria that had been implicitly used by the musicologists. We focused on using local features in our viewpoint representations, since van Kranenburg et al. (2013) showed that local features,

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: $$int0(p_0) = p_0$$ $$int0(p_n) = (p_n.onset,\ p_n.pitch - p_0.pitch)$$
pp	A string of (*onset, pitch pointer*) points: $$pp(p_0) = p_0$$ $$pp(p_n) = \begin{cases} (p_n.onset,\ p_n.pitch), & \text{the first time the pitch occurs; and} \\ (p_n.onset,\ j - n), & \text{otherwise;} \end{cases}$$ where j is the index of the most recent occurrence of the pitch $p_n.pitch$.
ioi	Inter-onset interval: $$int(p_0) = p_0$$ $$ioi(p_n) = (p_n.onset - p_{n-1}.onset,\ p_n.pitch)$$
oip	Same as *pp* but for onset-intervals: $$oip(p_0) = p_0$$ $$oip(p_n) = \begin{cases} (p_n.onset - p_{n-1}.onset,\ p_n.pitch), & \text{the first time the IOI occurs; and} \\ (j - n,\ p_n.pitch), & \text{otherwise;} \end{cases}$$ where j is the index of the most recent occurrence of the IOI, $p_n.onset - p_{n-1}.onset$.

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

such as motivic similarity, performed better than global features such as key, median and first/last note.

It is feasible that our results could have been improved by using higher-level structural information in our viewpoints such as the metrical positions of event onsets or the tonal functions of notes within keys (e.g., by using a pitch encoding that includes scale degree information). Unfortunately, such metrical and tonal information was not provided explicitly in the input data and would thus have had to have been either generated automatically or manually encoded. Moreover, using only low-level, "surface" information (e.g., note onsets and pitches) as input to our classifiers simulates more closely the information with which a listener is provided when recognizing the tune family of a melody without having studied a (transcribed) score of that melody (note that these melodies were only relatively recently written down after having been transmitted orally for generations). Of course, when hearing the melodies, a listener is very likely to infer a metre and a key at each point in the music, relative to which pitched events are interpreted. However, such higher-level metric and tonal information is *inferred* by a listener's brain (potentially drawing on all of that listener's musical knowledge) and is typically not *explicitly* encoded in the *physical* sound that impinges on the listener's ears. By restricting the information given as input to the classifiers to low-level information about the pitches and onsets of notes, we ensure that the task that we demand of our classifiers more closely resembles that which is carried out by the musicologists who created the ground-truth classification. While we accept that note onsets and pitches are also aspects of the experience of listening to a melody that are inferred by a listener's brain, we contend that there is rather less room for disagreement between listeners regarding what the pitches and onsets of notes are in a melody, than there is regarding higher-level structures such as metre and tonality. We therefore avoided using such higher-level structural information in the representations used by our classifiers, in order to minimize the risk of these classifiers depending on specific interpretations of the melodies that might not be shared by most listeners.

If such high-level information had been manually encoded in the input data by experts, then we could perhaps have reasonably assumed that this information had some legitimacy, but there would still have been the possibility of an expert encoding a metrical or tonal structure that reflected an idiosyncratic, theory-laden or controversial interpretation of the melody. On the other hand, if these structures had been generated automatically, then we could not have guaranteed that they reflected *anyone's* interpretation of the music. Moreover, the results would then have depended on the specific algorithms used to generate the higher-level structures, which would have made it much harder to assess the contributions made by the different compression algorithms.

Notwithstanding these arguments, we did, in fact, use *morphetic pitch* (Meredith, 1999, 2006, 2007; Collins, 2011) rather than chromatic pitch (or MIDI note number) in all of our experiments. As explained by Meredith (2006, pp. 127), the morphetic pitch of a note is an integer that is determined by the vertical position of the note-head of the note on the staff, the clef in operation on that staff at the location of the note and the transposition of the staff. Moving a note one step up on the staff (while keeping the clef constant) increases its morphetic pitch by 1, regardless of the note's accidental. The morphetic pitch of A0 is defined to be 0, thus A0, A♭0 and A♯0 all have a morphetic pitch of 0. The morphetic pitch of middle C (and C♯4, C♭4 and so on) is 23. Note that it is possible for a note to have a higher chromatic pitch but lower morphetic pitch than another note. For example, B𝄪3 has a lower morphetic pitch (22) but a higher chromatic pitch than C4. If p_{m} is the morphetic pitch of a note, then the *continuous name code* of the note in Brinkman's (1990, p. 126) system of pitch representation is $p_{\mathrm{m}} + 5$ and the *diatone* of the note in Regener's (1973, p. 32) system is $p_{\mathrm{m}} - 17$. For a more detailed discussion of morphetic pitch, chromatic pitch and other pitch representations, see Meredith (2006, pp. 126–130).

In a two-dimensional point-set representation, such as the ones that we employed, in which the first co-ordinate gives the onset time of a note and the second gives its morphetic pitch, patterns of notes related by modal transposition (e.g., (C,D,E) being transposed up a third within a C major scale to (E,F,G)) are translationally equivalent (i.e., they have the same shape). Such patterns are therefore discovered by algorithms like COSIATEC that detect transposition- (or translation-) invariant occurrences. They can also be discovered by general-purpose compression algorithms like LZ77, if the input encoding represents intervals between consecutive melodic notes rather than the notes themselves (as in our *int* and *pp* representations, see Table 1). It should be noted (again notwithstanding our arguments above) that the morphetic pitch values of the notes in our input data were *computed* using the PS13s1 pitch-spelling algorithm (Meredith, 2006, 2007). However, unlike metrical and tonal analysis algorithms whose output can be quite controversial, the output of the PS13s1 algorithm has been shown to reliably generate output that corresponds almost perfectly to the way that musical experts spell pitches in tonal and modal music. This, incidentally, provides evidence for there being something of a consensus among experts as to how pitches should be spelt in tonal and modal music, in contrast to, for example, key and harmonic structure, over which experts commonly disagree.

An important advantage of the representations chosen for this study is that they result in a considerable amount of redundancy. Indeed, if the onsets had not been suitably transformed, all notes would have mapped to distinct symbols, resulting in strings that could not have been compressed using the general-purpose compressors tested here. As already noted, our representations also allow for the discovery of patterns related by transposition (both modal and, at least in most cases in tonal music, chromatic).

To recap, in our experiments reported below, each melody was represented as a string of two-dimensional points, (t, p), each representing a note, such that t is the *onset time* of the note and p is the *morphetic pitch* of the note. Unless otherwise stated, all representations are applied to strings in which these (t, p) points have been sorted into lexicographic order.

We define a *compressed viewpoint* to be a pair, (Z, R), where Z is a compression algorithm and R is a viewpoint. A compressed viewpoint can be seen as a function, $Z \circ R$, that takes a melody in the *pitch-time* representation and returns a string of symbols forming the encoding of that melody from that compressed viewpoint.

3.2 Normalized compression distance

As already mentioned above, *normalized compression distance* (NCD) (Li et al., 2004) has been used as a measure of similarity between melodies in a number of previous studies (Cilibrasi et al., 2004; Hillewaere et al., 2012; Li and Sleep, 2004, 2005; Meredith, 2014a,b, 2015, 2016). Normalized compression distance is a practical proxy for *normalized information distance*, an ideal similarity metric, based on the Kolmogorov complexity of an object, which is (roughly speaking) the length in bits of the shortest program that generates the object as its only output. Li et al. (2004) defined the *normalized information distance* (NID) between two objects x and y, as follows:

$$d(x, y) = \frac{\max\{K(x|y^*), K(y|x^*)\}}{\max\{K(x), K(y)\}}\ , \tag{4}$$

where $K(x)$ is the Kolmogorov complexity of x and $K(x|y^*)$ is the conditional complexity of x given a description of y whose length is equal to the Kolmogorov complexity of y. But as the Kolmogorov complexity cannot, in general, be computed, it has to be estimated by the length of a real compressed object. Therefore, Li et al. (2004) proposed the *normalized compression distance* (NCD) as an estimator of the NID. Here, NCD is defined for a compressed viewpoint, (Z, R), and two melodies, s and s', as follows:

$$\mathrm{NCD}(Z, s, s') \ = \ \frac{|Z(ss')| - \min\{|Z(s)|, |Z(s')|\}}{\max\{|Z(s)|, |Z(s')|\}}\ , \tag{5}$$

where Z is a real-world compressor (e.g., LZ77), $|x|$ is the length of encoding x and ss' is the concatenation of melodies s and s'.

3.3 Corpus compression distance

Unfortunately, the distance defined in Eq. 5 has two problems. First, the values are not restricted to being in the interval $[0, 1]$. Second, for two different compression 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 devised a new distance measure, which we call *Corpus Compression Distance* (CCD), that depends not only on the compression algorithm, Z, but also the corpus, C, of *labelled* melodies used for classification. This novel measure has the feature that it computes values in the interval $[0, 1]$ for all algorithms. If our task is to label a melody, s, then we find the distance from s to each labelled melody, s', in C using the CCD, which is defined as follows:

$$\mathrm{CCD}(s, s', Z, C) \;=\; \frac{\mathrm{NCD}(Z, s, s') - \min(\mathcal{D}(s, C))}{\max(\mathcal{D}(s, C)) - \min(\mathcal{D}(s, C))}\;, \tag{6}$$

where $\mathcal{D}(s, C) = \{\mathrm{NCD}(Z, s_1, s_2) \mid s_1, s_2 \in C \cup \{s\}\}$ and $\min(\mathcal{D}(s, C))$ and $\max(\mathcal{D}(s, C))$ are, respectively, the minimum and maximum values in the set, $\mathcal{D}(s, C)$. To evaluate the algorithms, we also examined the compression factors achieved, since these appeared to be related to the classification success rates. The *compression factor*, $CF(v, s)$, achieved by an algorithm that generates an encoding, v, for a melody, s, is defined by:

$$CF(v, s) = \frac{|s|}{|v|}\;. \tag{7}$$

Finally, the classification success rate is defined as follows:

$$SR = \frac{\text{number of correctly classified melodies}}{\text{number of melodies in the corpus}}\;. \tag{8}$$

3.4 Classification Method

The classification method takes a melody and a corpus as input and aims to return a class which is 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:

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.

- 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 be in j. It can be seen as a non-normalized estimation of the conditional probability defined by Conklin (2013a,b), that is $P(j|s, v)$. But for this estimation, the method computes a score depending on nearest neighbours instead of n-grams. The value of f is given by the following formula,

$$f(C, s, j, v, N) = \sum_{s_i \in C_j^N(s)} \frac{1}{(\mathrm{CCD}(s, s_i, v, C) + \epsilon)^{N_i}} \ , \tag{9}$$

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

$$C_j^N(s) \ = \ C_j \cap C^N(s) \ , \tag{10}$$

where C_j is the subset of C which contains the melodies in class, j, and $C^N(s)$ contains 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 $\epsilon = 0.1$. N_i is the index assigned to the nearest neighbour s_i—that is, $N_i = N - i + 1$.

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,

$$g(j) \;=\; \frac{|C_j|}{|C|} \sqrt[n]{\prod_{i=1}^{n} M_{i,j}} \;, \tag{11}$$

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:

$$c^* \;=\; \operatorname*{argmax}_{c\in[1,m]} g(c) \;. \tag{12}$$

4 Results

The algorithms described above were first evaluated on the task of classifying melodies in the *Annotated Corpus* from the Dutch Song Database (`http://www.liederenbank.nl`). LZ77 and COSIATEC were then compared on the task of discovering subject and counter-subject 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.

4.1 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, 2008). 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 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 3 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 the LZ77 parameters were chosen.

As explained in Section 3.1, the pitch of each note in the input representations was represented by its morphetic pitch, computed from the MIDI data using the PS13s1 algorithm (Meredith, 2006, 2007).

4.1.1 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. We used the representations defined in Table 1 above.

As all melodies in this corpus are monophonic, the onset times of the notes in a melody are all distinct. A consequence of this is that, if the basic representation is used (see Table 1), every symbol is distinct, leading to no repeated substrings, which results in the general-purpose text-compression algorithms being unable to find any repeated patterns. These algorithms can therefore only work on representations that transform the onset values. Note that this problem does not apply to COSIATEC.

Conversely, COSIATEC cannot use representations that transform the onsets (*ioi*, *oip* and combined—see Table 1). Those representations worked well for LZ77, LZ78 and 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. We tried solving this problem by adding a third dimension corresponding to the index of a note, but this drastically reduced the performance of the algorithm, both in terms of classification (less than 70%) and compression (some compression factors were less than 1). Therefore, all COSIATEC compressed viewpoints that involved transforming onsets were discarded.

Table 3 shows the results obtained by using the classification method on each compressed viewpoint separately (i.e., in each case, the table corresponding to Table 2 contained only one row). Only those compressed viewpoints that resulted in a success rate higher than 70% are listed (along with the highest-scoring compressed viewpoint for LZ78). Moreover, when the compression factor achieved by a particular compressed viewpoint, (Z, R), was less than 1, this was invariably associated with a poor classification success rate, so all compressed viewpoints with an average compression factor less than 1 were discarded. We can see in Table 3 that, in terms of success rate, the combination of COSIATEC with the basic (onset, morphetic pitch) representation outperformed all of the other compressed viewpoints with a classification success rate of 0.8528. The compressed viewpoint (COSIATEC, *int*) achieved poorer results than (COSIATEC, *basic*), implying that the patterns found were not the same with both representations. Therefore, it is very important to find the representation that provides the best success rate for a given compression algorithm.

Compressed viewpoint	*1-NN Leave-one-out SR*	CF_{AC}	CF_{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, *oip*)	0.7944	1.1188	1.2629
(LZ77, $int0 \circ oip$)	0.7861	1.1806	1.3306
(COSIATEC, *int*)	0.7556	1.5266	1.6226
(LZ77, $ioi \circ oip$)	0.7472	1.0088	1.1127
(LZ77, $int \circ oip$)	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, $int0 \circ oip$)	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; CF_{AC} denotes mean compression factor on *Annotated Corpus*; and CF_{pairs} is the mean compression factor on pair files used to compute the NCDs.

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 compression factor 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).

Figure 5 shows graphs of compression factor against success rate for the values in Table 3. In each case there was a weak, insignificant, negative correlation, indicated by the trend lines (for CF_{AC}: $r = -0.4221, N = 14, p = 0.133$; for CF_{pairs}: $r = -0.4144, N = 14, p = 0.141$). It is important to note, however, that Table 3 only shows values of compression factor and success rate for the best-performing compressed viewpoints. The fact that no significant correlation was found for this particular collection of relatively well-performing viewpoints does *not* imply that there is no correlation between compression factor and success rate *in general*. Recall that, as explained above,

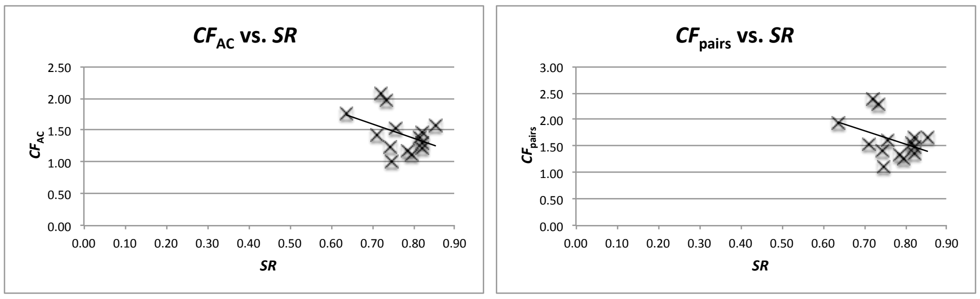

Figure 5: Graphs of compression factor (CF) against success rate (SR) for the values in Table 3. The graph on the left shows the mean compression factors on the *Annotated Corpus* (i.e., with each melody compressed individually)(CF_{AC}); the graph on the right shows the mean compression factors for pairs of concatenated melodies (CF_{pairs}).

all viewpoints resulting in poor compression (i.e., with mean compression factors less than 1) were discarded because they were also invariably associated with poor success rates.

4.1.2 Combined Viewpoints Classification

Having tested the algorithms with single compressed viewpoints, we then carried out an evaluation in which the best compressed viewpoints were combined. We chose to use the combined representations method only on compressed viewpoints that gave good results when used alone. We then tested different combinations to determine which compressed viewpoints improved the result. Table 4 shows the success rates obtained by the combined representations method using the n compressed viewpoints that performed best individually. All these results are better than those obtained using single compressed viewpoints (cf. Table 3). However, it seems that some compressed viewpoints have a detrimental effect on success rate (e.g., (LZ77, $int0 \circ ioi$), (COSIATEC, int)). The last result in Table 4, denoted by $10'$, is obtained by combining eight of the ten best compressed viewpoints, omitting (LZ77, $int0 \circ ioi$) and (COSIATEC, 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 factor. 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 0.9444.

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

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.

speculate that the difference may be due to the fact that he was additionally using duration and metrical information to build viewpoints, while we only use pitch and note onset information.

4.2 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: $int \circ ioi$, $int \circ oip$, ioi, $int0 \circ ioi$, $ioi \circ ioi$, oip, $int0 \circ oip$ and $ioi \circ oip$. 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 (see Eq. 3 above), 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.

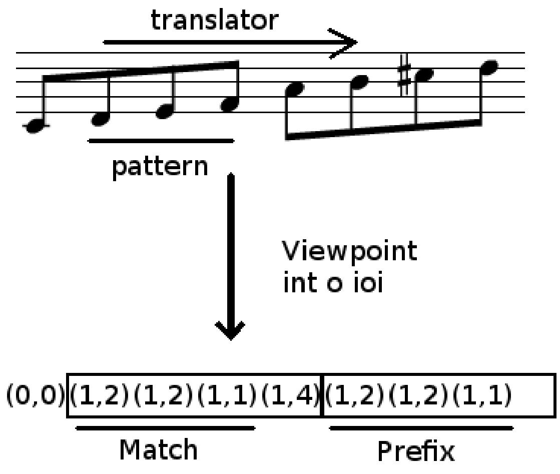

Figure 6: One iteration of LZ77, modified for pattern discovery.

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 6 shows an example of an iteration of the first of these two steps. The match is done on the second representation which is *int* ∘ *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 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 (TLF1) (see Meredith (2015, pp. 256–259) 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 factor 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 a special subclass of 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 other hand, LZ77 only discovers maximal sub*strings*, which implies that

Compressed viewpoint	*TLF1*	*TLP*	*TLR*	*CF*
(COSIATEC, *basic*)	0.123	0.071	0.523	2.6404
(LZ77, $int \circ ioi$)	0.053	0.035	0.125	2.1976
(LZ77, $int \circ oip$)	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, *oip*)	0.044	0.030	0.087	1.3929
(LZ77, $int0 \circ oip$)	0.044	0.030	0.087	1.4255
(LZ77, $ioi \circ oip$)	0.043	0.030	0.080	1.3374

Table 5: Results for the pattern discovery task on Bach's fugues with the compressed viewpoints for 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 F_1 score and *CF* is average compression factor.

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 *TLF1* 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 $int \circ ioi$ representation, under these input data sorting conditions, LZ77 outperformed COSIATEC substantially in terms of compression factor and also marginally in terms of *TLF1* and *TLP*. 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 factor achieved seems to be related to pattern discovery performance. Indeed, there is a strong, highly significant, positive correlation, between compression factor and TLF1 both for the data in Table 5 ($r = 0.8606, N = 9, p = 0.003$) and for the data in Table 6 ($r = 0.8542, N = 9, p = 0.003$)

Compressed viewpoint	*TLF1*	*TLP*	*TLR*	*CF*
(LZ77, $int \circ ioi$)	0.124	0.073	0.521	3.7142
(COSIATEC, $basic$)	0.123	0.071	0.523	2.6404
(LZ77, $int \circ oip$)	0.120	0.072	0.441	2.5008
(LZ77, ioi)	0.114	0.073	0.298	1.9374
(LZ77, $int0 \circ ioi$)	0.114	0.073	0.298	1.9553
(LZ77, $ioi \circ ioi$)	0.108	0.068	0.280	1.7152
(LZ77, oip)	0.100	0.065	0.234	1.5428
(LZ77, $int0 \circ oip$)	0.100	0.065	0.234	1.5584
(LZ77, $ioi \circ oip$)	0.091	0.063	0.202	1.3424

Table 6: Results obtained on the pattern discovery task on Bach's fugues using various compressed viewpoints based on LZ77 and COSIATEC with the basic representation. Results are 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 *CF* are defined as in Table 5.

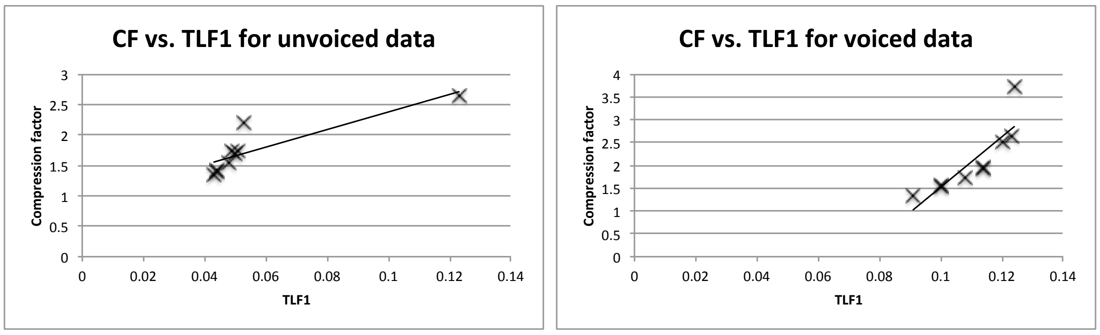

Figure 7: Graphs of compression factor (CF) against three-layer F1 score (TLF1) for the values in Table 5 (left) and Table 6 (right). CF is strongly positively correlated with TLF1 in both sets of data, as indicated by the linear trend lines. See main text for details.

(see also the graphs in Figure 7). This supports the hypothesis, suggested by Kolmogorov complexity (Kolmogorov, 1965) and the minimum description length principle (Rissanen, 1978), that shorter descriptions represent better analyses. However, the results on this pattern discovery task also illustrate the importance, both for compression and pattern discovery, of choosing a data representation that is appropriate for a given compression algorithm.

5 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 F_1 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 hypothesized 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 caused the notes to be re-ordered so that contiguous sequences of notes in voices in the fugues 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 F_1 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 factor achieved by an algorithm and its pattern-discovery performance (measured in terms of three-layer F_1 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. However, the usefulness and performance of such algorithms depend critically on the input data being represented appropriately.

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.

$$\cdots abc\ \boxed{defghijk}\boxed{abcdefgh}\ ijk\cdots$$

Figure 8: Worst case of LZ77 algorithm: periodic input of period p with $p > n - L_s$.

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 symbols. As is shown in Figure 8, the algorithm cannot find the first symbol in the dictionary and consequently cannot find any pattern. It is therefore important to find good parameters which correspond to the analysed corpus.

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 $(\mathrm{LZ77}_{n,L_s}, \mathit{int} \circ \mathit{ioi})$. Each pair of parameters (n, L_s) produced a success rate, an average compression factor on melody files, and an average compression factor 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 factors. Moreover, the last line shows poor compression factor but a high classification success rate. Poor compression factors 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 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

n	L_s	*1-NN Leave-one-out SR*	CF_{AC}	CF_{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 ($\text{LZ77}_{n,L_s}$, $int{\circ}ioi$). *SR* is success rate; CF_{AC} is the mean compression factor on the *Annotated Corpus*; and CF_{pairs} is the average compression factor on pair files used to compute the CCDs.

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 once in a piece, reducing the size of the dictionary will not decrease the compression factor 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

Bimbot, F., Deruty, E., Sargent, G., and Vincent, E. (2012). System & Contrast : a Polymorphous Model of the Inner Organization of Structural Segments within Music Pieces (Original Extensive Version). Research Report IRISA PI-1999, IRISA.

Brinkman, A. R. (1990). *PASCAL Programming for Music Research.* The University of Chicago Press, Chicago, IL.

Burrows, M. and Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm. Technical Report SRC 124, Digital Systems Research Center (now HP Labs), Palo Alto, CA.

Chordia, P., Sastry, A., Mallikarjuna, T., and Albin, A. (2010). Multiple viewpoints modeling of tabla sequences. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, Utrecht, The Netherlands.

Cilibrasi, R., Vitányi, P. M. B., and de Wolf, R. (2004). Algorithmic clustering of music based on string compression. *Computer Music Journal*, 28(4):49–67.

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

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). `http://www.algomus.fr/truth/fugues.truth.2013.12`.

Grijp, L. P. (2008). 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, Amsterdam, NL.

Hillewaere, R., Manderick, B., and Conklin, D. (2009). Global feature versus event models for folk song classification. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR 2009)*, pages 729–733, Kobe, Japan.

Hillewaere, R., Manderick, B., and Conklin, D. (2012). String methods for folk tune genre classification. In *Proceedings of the 13th International Society for Music Information Retrieval Conference (ISMIR 2012)*, pages 217–222, Porto, Portugal.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Problemi Peredachi Informatsii*, 1(1):3–11.

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

Li, M. and Sleep, R. (2004). Melody classification using a similarity metric based on Kolmogorov complexity. In *Proceedings of the Sound and Music Computing Conference (SMC '04)*, Paris, France.

Li, M. and Sleep, R. (2005). Genre classification via an LZ78-based string kernel. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)*, pages 252–259, London, UK.

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. `http://www.mattmahoney.net/dc/text.html` (Last accessed 11 August 2015).

Mavromatis, P. (2005). A hidden markov model of melody production in Greek church chant. *Computing in Musicology*, 14:93–112.

Mavromatis, P. (2009). Minimum description length modelling of musical structure. *Journal of Mathematics and Music*, 3(3):117–136.

Meredith, D. (1999). The computational representation of octave equivalence in the western staff notation system. In *Cambridge Music Processing Colloquium*, Department of Engineering, University of Cambridge. `http://www.titanmusic.com/papers/public/meredith.pdf`.

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

Meredith, D. (2007). *Computing Pitch Names in Tonal Music: A Comparative Analysis of Pitch Spelling Algorithms*. PhD thesis, Faculty of Music, University of Oxford.

Meredith, D. (2012). Music analysis and Kolmogorov complexity. In *Proceedings of the 19th Colloquio di Informatica Musicale (XIX CIM)*, Trieste, Italy. Available online at `http://www.titanmusic.com/papers/public/cim20121_submission_105.pdf`.

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. `http://dx.doi.org/10.1109/CIP.2014.6844503`.

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. (2015). Music analysis and point-set compression. *Journal of New Music Research*, 44(3):245–270.

Meredith, D. (2016). Analysing music with point-set compression algorithms. In Meredith, D., editor, *Computational Music Analysis*, pages 335–366. Springer, Cham, Switzerland. `http://link.springer.com/chapter/10.1007/978-3-319-25931-4_13`.

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*, Department of Engineering, University of Cambridge. `http://www.titanmusic.com/papers/public/cmpc2003.pdf`.

Nattiez, J.-J. (1975). *Fondements d'une sémiologie de la musique.* Union Générale d'Editions, Paris.

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

Pavlov, I. (2015). 7-Zip. `http://www.7-zip.org`. Last accessed 11 August 2015.

Pearce, M., Conklin, D., and Wiggins, G. (2005). Methods for combining statistical models of music. In Wiil, U. K., editor, *Computer Music Modeling and Retrieval*, volume 3310 of *LNCS*, pages 295–312. Springer.

Regener, E. (1973). *Pitch Notation and Equal Temperament: A Formal Study.* University of California Press, Berkeley, CA.

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

Ruwet, N. (1966). Méthodes d'analyses en musicologie. *Revue belge de musicologie*, 20(1/4):65–90.

Sayood, K. (2012). *Introduction to Data Compression.* Morgan Kaufmann, Waltham, MA., 4th edition.

Seward, J. (2010). bzip2 version 1.0.6, released 20 September 2010. `http://www.bzip.org` (Accessed 19 April 2014).

Shannon, C. E. (1948a). A mathematical theory of communication (Part I). *The Bell System Technical Journal*, 27(3):379–423.

Shannon, C. E. (1948b). A mathematical theory of communication (Part II). *The Bell System Technical Journal*, 27(4):623–656.

Solomonoff, R. J. (1964a). A formal theory of inductive inference (Part I). *Information and Control*, 7(1):1–22.

Solomonoff, R. J. (1964b). A formal theory of inductive inference (Part II). *Information and Control*, 7(2):224–254.

Temperley, D. (2014). Probabilistic models of melodic interval. *Music Perception*, 32(1):85–99.

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.

Vitányi, P. M. and Li, M. (2000). Minimum description length induction, bayesianism, and Kolmogorov complexity. *IEEE Transactions on Information Theory*, 46(2):446–464.

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

White, C. W. (2014). Changing styles, changing corpora, changing tonal models. *Music Perception*, 31(3):244–253.

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. *IEEE Transactions on Information Theory*, 24(5):530–536.