COMPRESSION-BASED DEPENDENCIES AMONG RHYTHMIC MOTIFS IN A SCORE

Pierre Donat-Bouillud[1], Samer Abdallah[2], Nicolas Gold[2]

[1] *ENS Rennes*

[2] *Department of Computer Science, University College London*

Correspondence should be addressed to: s.abdallah@ucl.ac.uk

Abstract: **Music similarity has been widely studied through melodic and harmonic matching, clustering, and using various metrics for measuring distance. Such analyses offer the musicologist a view of the 'sameness' of parts of a score. However, similarity alone does not necessarily allow exploitation of that sameness in reasoning about the music. In this paper, we present work in progress to investigate rhythm similarity at various scales, beginning at the smallest (single measures or groups of measures). We use normalised compression distance and variations thereof to derive similarity-based dependencies between parts of the music. Establishing such dependencies may allow software engineering dependence analysis techniques to be applied to music to, e.g. remove from focus aspects not relevant to a particular enquiry ('slicing'), determine the sensitivity of later parts of the music on former parts ('impact analysis'), and to find motivic processes and developments within the musical form.**

The analysis will thus draw on software engineering techniques, information theory, and data compression. Our results thus far show that text-based compressors introduce significant non-linear artefacts at small scales making similarity identification based on compressed lengths difficult. Future work will involve progressively larger scale music to determine the sensitivity of the results to the size of music being analysed in order to guide musicologists wanting to adopt similar approaches. We expect to find that at larger scales, the artefacts in text compression become less significant and identifying the threshold at which this happens is thus important. We discuss tree compression as having the potential to capture musically-important relationships lost by text compression and believe that this approach would be more successful at small scales.

1. INTRODUCTION

One of the basic components of music analysis is the detection of patterns, which can be imprinted on the music in a seemingly limitless number of ways: patterns of duration, pitch, harmony, timbral, patterns that repeat exactly, patterns that do not repeat but develop in a systematic way, patterns that are translated, dilated, reversed or otherwise transformed, patterns that are easy to spot, patterns that are hard to spot, and so on. That this should be so is perhaps not surprising, given that the full weight of human intelligence and ingenuity can find expression in music. But what, in this context, do we mean by 'pattern'?

The information-theoretic concept of *redundancy* is a good candidate for a general definition 'pattern'—it involves, essentially, any regularity, any departure from complete unpredictability. In *algorithmic* information theory [1], the presence of patterns implies *compressibility*: the idea that an object is amenable to a description which is shorter than the original object.

In this paper, we describe our preliminary and ongoing research into the use of compression-based approximations to *Kolmogorov complexity*—a measure of the amount of information in an object—and how this can be used to define a general concept of similarity. We restrict the analysis to purely rhythmic data (i.e., information about note onset times and durations). We first present the theoretical underpinnings of algorithmic information theory and then apply these to rhythmic data in §3. Our approach is similar to that of Cilibrasi et al [2], but whereas they investigated similarity measures between entire pieces, we examine similarity between relatively short *fragments*. This places unusual demands on the compression algorithms, as they are not designed for compressing small objects. The paper concludes by identifying directions for future work, discussing tree compression in particular as a potential route to small-fragment compression.

2. ALGORITHMIC INFORMATION THEORY

A central concept in algorithmic information theory (AIC) [3] is that a computer program, when run, can produce an output which is much 'larger', in the sense of numbers of symbols, or elements in some data structure, than the original program. This suggests that the amount of information in an object x is upper-bounded by the amount of information in a program P that produces x as output. The program can be thought of as a 'compressed' version of the output, since it is able to reproduce the output exactly, given a suitable interpreter of the language, which itself is a program of a certain fixed size. If the program P can be further compressed (for example, using a standard compression method such as Zip, or BZ2), then the size of the compressed program, becomes an upper-bound on the amount of information in the original object x.

In this way, the *Kolmogorov complexity* $K(x)$ of an object x is defined as the length of the shortest program that produces x as output. For definiteness, we choose to represent x and any program as a string binary digits, so that the length of the string gives the amount of information in bits. The length of the shortest program will depend on what programming language we use, but for any two languages L_1 and L_2, if L_2 is *Turing complete*, then an interpreter of L_1 can be written in L_2, which will be of a fixed length independent of x. Thus, if we are interested in comparing the complexities of several objects with respect to a fixed language, then the choice of that language will only affect their complexities up to an (unknown) additive constant. Such additive constants appear often in AIC, and mean that the theorems need to be interpreted with care in practical applications. In this paper, we will simply use approximate equality ($\approx$) to represent such relations, and refer to reader to [1] for a more detailed analysis.

An additional difficulty with applications of Kolmogorov complexity is that there is no general purpose algorithm for finding the shortest program for a given x, or indeed its length. The best that can be done is to find *some* program that produces x and use its length as an upper bound on $K(x)$, but this does not guarantee that a shorter program could not be found. In practice, the search for a short description can be done by a compression program—the compressed output can be thought of as a 'program' in a language determined by the compression algorithm.

This approach of using a compression algorithm links AIT with Shannon's information theory [4] and probabilistic modelling: it is known that optimal compression of a certain class of object requires an understanding of the *probabilistic* structure of the relevant domain; that is, a compressor designed to compress an arbitrary x drawn from some domain $\mathscr{X}$ can do better on average by knowing the probability that it will be asked to compress a particular x. In this way, good compression is closely linked with good probabilistic modelling. For example, knowing the relative frequencies of word use in a language means that shorter codes can be used for more frequently-occurring words.

When the probabilistic structure of a domain is unknown *a priori*, (for example, as in a single piece of music), then good compression depends on using an algorithm which learns about the patterns in the object as it compresses: this is what general purpose compression algorithms do. They are examples of 'universal compressors', which means that given a sufficiently large object, they are able

to compress as well as is possible to. However, while this *asymptotic* optimality is a desirable theoretical property, it does not guarantee that any given universal compressor will perform well for given finite object. This is in accord with the 'no free lunch' theorems of machine learning [5], which state that no single adaptive probabilistic model is guaranteed to fit all datasets better than any other, and hence there is no single 'best' compression algorithm.

2.1. Information distances

The algorithmic mutual information (AMI) is provides a measure of the information shared between two objects and hence can form the basis for a measure of similarity. It is defined for two objects x and y as

$$I(x;y) = K(x) + K(y) - K(x,y). \quad (1)$$

The joint Kolmogorov complexity $K(x,y)$ is the length of the shortest program that produces the two outputs x and y. The mutual information, then, quantifies the idea that 'knowing x helps you compress y', or vice-versa.

Alternative expressions for the AMI involve the *conditional* Kolmogorov complexity $K(y|x)$, which is the length of the shortest program that, taking x as input, produces y. In these terms, we can also write

$$K(x,y) \approx K(x) + K(y|x) \approx K(y) + K(x|y) \quad (2)$$

and hence

$$I(x;y) \approx K(x) - K(x|y) \approx K(y) - K(y|x). \quad (3)$$

Though the AMI can reasonably be understood as a measure of similarity, it cannot be interpreted as a *distance* (or metric) in the mathematical sense. For, example, if $x = y$, then $I(x;y) \approx K(x)$ (since $K(x,x)$ is just $K(x)$ plus a small constant to indicate that x should be output twice instead of once), but if x and y become 'larger', then $I(x;y)$ generally becomes larger too, even if x and y are similar. [6] propose several 'information distances' which are metrics, including

$$E_1(x,y) = \max\{K(x|y), K(y|x)\}. \quad (4)$$

Vitanyi *et al* [7, 8] subsequently proposed the normalised information distance (NID) which is approximately invariant with the absolute size of the objects being compared and is defined as

$$d_{\mathrm{NID}}(x,y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}. \quad (5)$$

2.2. Compression and compression distance

Since the Kolmogorov complexity is not computable, we must, for practical applications, find an approximation. In the absence of more specific knowledge about the class of objects to be compressed, a reasonable start is to use a general purpose compression algorithm. These have the advantage that some have certain general asymptotic optimality (so called *universality*) properties.

If we let $C(x)$ be the length of the binary string produced by such a compression program given and object x, also encoded as a binary string, then the *normalised compression distance* (NCD) [9, 10] is

$$d_{\mathrm{NCD}}(x,y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}, \quad (6)$$

where xy is the binary string resulting from the concatenation of the binary strings representing x and y. Note that this is an approximation: it may not even be symmetric, since a given compressor might yield $C(xy) \neq C(yx)$.

In this work, we examined the behaviour of four well-known compression programs when compressing small rhythmic patterns: **LZO** is a block compression algorithm belonging to the LZ77 (Lempel-Ziv) family. **Snappy** also belongs to the LZ77 family, and is optimised for speed of compression. **Zlib** is based on the DEFLATE algorithm, which combines LZ77 with Huffman encoding. **BZ2** is based on the Burrow-Wheeler transform and Huffman encoding.

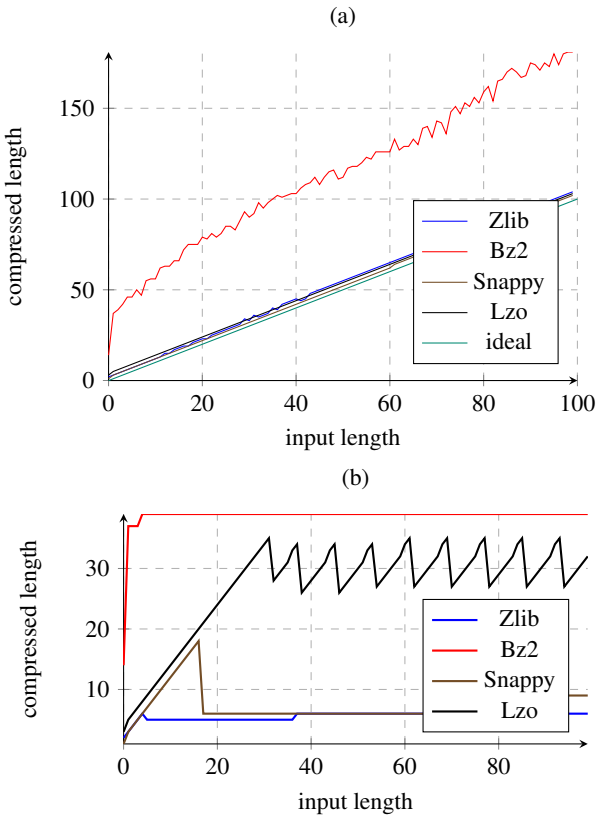


Figure 1: Size of (a) random strings and (b) repeated characters after being compressed by several algorithms: zlib, bz2, snappy, and lzo. For a given input size and class, all algorithms process the same string. As the strings in (a) are random, a perfect compressor should not be able to compress the strings, yielding a line of slope 1 (green line). In (b), the ideal performance is not well defined unless we specify a probability distribution over the length of the string.

2.3. Compression artefacts

Intuitively, one would expect that the size of a compressed object would increase smoothly and monotonically with the size of the original object, but for the compressors we examined, this behaviour is realised only approximately, with departures from this ideal especially apparent for small objects, where overheads such as coding dictionaries (the fragments of text or music observed during compression) may dominate. Fig. 1 illustrates how the four general purpose compressors perform on a string of independent identically distributed pseudorandom symbols.

3. Similarities between rhythmic fragments

3.1. Similarities between measures

As an initial experiment, we took music scores (in MusicXML format), and represented short fragments of one or two measures of a single monophonic part as a sequences of characters with an arbitrary mapping of note durations to characters, e.g., *a* for a minim, *b* for a crotchet, *c* for a quaver, etc.. The NCD between these short sequences was then computed using each of the text compression programs. We omit detailed results here due to lack of space, but as the graphs of Fig. 1 suggest, the compression programs do not perform well on such short fragments and the similarity results did not appear to be meaningful.

3.2. Similarities between segments

To test the system on longer (but still short) segments of a few measures, we investigated the possibility that the NCD might guide the segmentation of a longer sequence into two parts, by finding

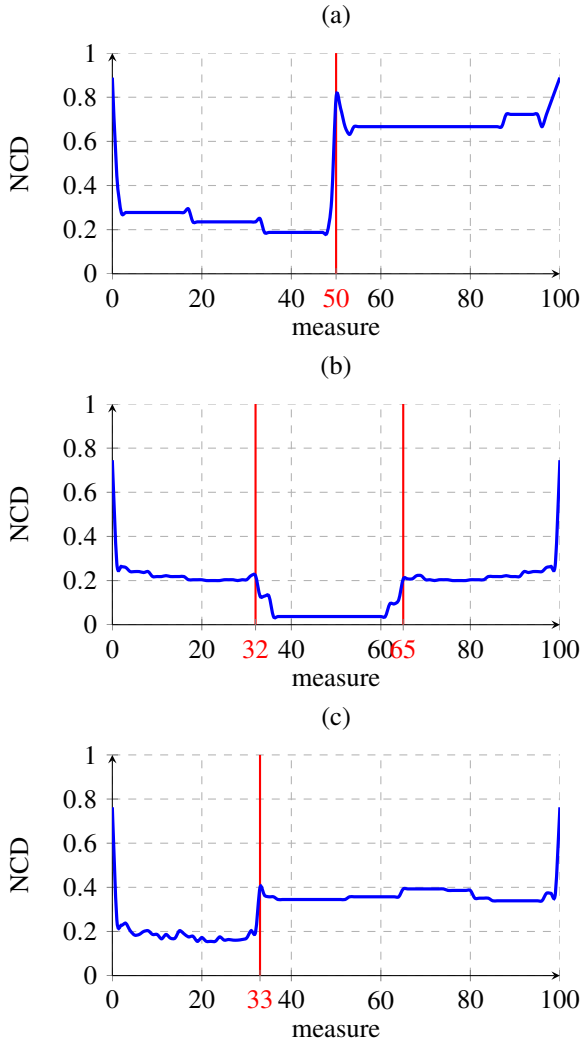


Figure 2: NCD using (a) zlib compressor with two different segments in the score; (b) bz2 compressor for a three-segment score.

a boundary such that the segments on either side are maximally dissimilar. The idea is that if the boundary is in the wrong place, then the presence of a certain pattern or rhythmic process on either side of the boundary with reduce the NCD between the two parts. Only when the boundary is in the right place, where there is an abrupt change in the nature of the rhythm, will the two parts have the least in common and thus have the largest NCD. This is similar to the idea of likelihood ratio testing for change-point detection. Thus, a tentative boundary is introduced at every point in the sequence and the NCD between the two segments x and y computed as a function of the split-point. We also compute the constituent terms of the NCD, $C(x), C(y)$ and $C(xy)$, as well as the estimated AMI $I(x;y)$. We expect the true boundary to be a local maximum of the NCD.

The idea was tested using artificially constructed rhythms; some results are illustrated in Fig. 2. For each of these three scores, two or three sections constructed by repeating a simple rhythmic phrase a number of times were concatenated. The score of Fig. 2(a) is

Although there are some artefacts which do not correspond to anything in the score, there is indeed a relatively pronounced local maximum in the NCD at the correct boundary position. The score for Fig. 2(b) consists of three sections:

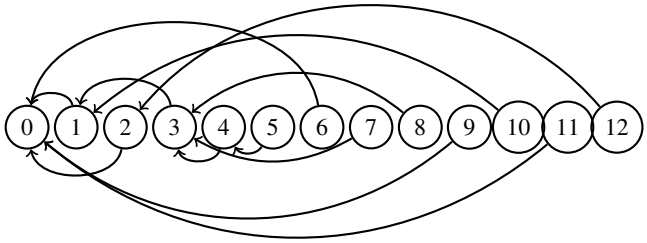
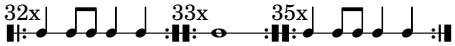
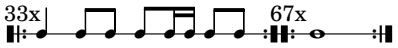


Figure 4: Backward dependencies for the same piece as in Fig. 3, using bz2. No threshold was used.



This score does not fulfil the assumption that there are only two segments, and does not have clear local maxima at the boundaries; nonetheless, the structure is somewhat reflected in the NCD graph. Finally, Fig. 2(c) was computed from a score with the same rhythmic patterns as Fig. 2(a), but with the boundary at about two-thirds of the way through:



3.3. Backward dependencies and slicing

Another possibility for using the NCD to aid clustering is to proceed through the score sequentially, taking windows of one or two measures at a time, and comparing each window with all of the previous windows to ascertain whether the current window is related to one of the previous ones, or introduces a new piece of thematic material. This could be seen as capturing a musical 'dependency' of sorts, potentially opening the possibility to apply dependence-based algorithms (e.g. static backward program slicing[12]: a technique for removing unrelated lines of source code in a program based on their relevance to a point of interest expressed by dependence) in music analysis. We tested an algorithm which for each new segment y, estimates the AMI $I(x;y) \approx C(x) - C(x|y)$ between y and each previous segment x and introduces a backward link from y to x for the $x = \arg\max_x I(x;y)$ which maximises the AMI, but only if $I(x;y)$ is larger than a certain threshold. Fig. 3 shows the result of this process for the opening of Mozart's Horn duet No.4 (K.V. 487) (part for Horn 1), using Zlib compression. Of the six links, four are reasonable (if somewhat trivial due to the identity of the rhythm in those segments). However, segment 10 is linked to segment 8, though we would expect it to be linked to segment 1. Fig. 4 shows the result of using bz2, with the linking threshold set to zero: here, segment 10 is linked to 1, but segment 7 is, unexpectedly, linked to segment 3. Determining why these unexpected links occur is the subject of future work.

4. Discussion and Conclusions

The notion of similarity is, in general, very broad and ill-defined: complex objects can be similar or dissimilar in many different ways. Domain specific measures of similarity can focus on arbitrary aspects of the objects being compared, and must be designed, sometimes in a rather ad-hoc fashion, for each new domain. Algorithmic information theory offers a more general answer to the question of similarity, by appealing to the concept of universal computation: if there is a pattern to be detected, then a universal computer will be capable of expressing it, so that whatever the basis of similarity between two objects, it should be detectable given a sufficiently 'universal' compression program. However, such 'universal' compression holds only asymptotically: when it comes to operating on real, finite data, the usefulness of compression distance depends on finding good, *domain specific* compression algorithms, which, Shannon's information theory tells us, depends on finding good probabilistic models of the data in question.

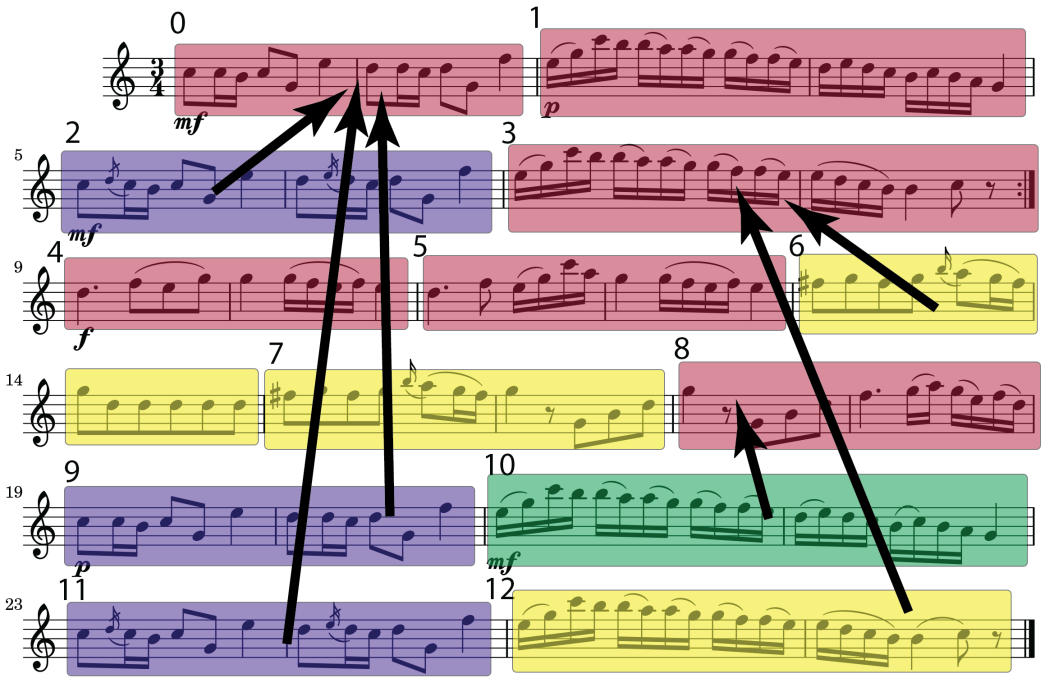


Figure 3: Backward dependencies for the Polonaise of Basset Horn Duets by Mozart, Horn 1 part (K.V. 487). All segments with the same color are linked to the same previous segment. The threshold was set to 4.

In our experiments, it appeared that general purpose text compression algorithms are not suitable for short fragments of symbolic music data.

Tree compression offers the potential to solve this problem (at least partly) for data which are amenable to tree representation. Since rhythmical groupings can be constructed as trees representing duration (with the root of each subtree representing the combined duration of its children), using the tree directly as the object to be compressed (rather than collapsing to a string representation where the compressor does not understand the hierarchical nature of the represented data) offers the potential to achieve better compression. Tree shapes can be encoded in the compression dictionary regardless of the scale of rhythm that they represent, creating a more efficient (and thus smaller) dictionary. We are implementing the tree compression algorithm of Chen and al. [13] with a view to repeating the experiments described above and comparing the results to the text compressors on small-scale data.

Other future work would include the application of slicing and other dependence-based algorithms to the dependence graph created by the similarity measures described here, and exploring how these could be used in a music-analytic context.

Acknowledgements This work was partially supported by the EPSRC CREST Platform grant [grant number EP/G060525/2] and the AHRC Digital Music Lab project [grant number AH/L01016X/1] and was undertaken while Pierre Donat-Bouillud was a visitor at UCL. Source code and data are available on request from Samer Abdallah.

References

[1] M. Li and P. M. B. Vitányi: *An introduction to Kolmogorov complexity and its applications*. Springer, 2009.

[2] R. Cilibrasi, P. Vitányi, and R. De Wolf: *Algorithmic clustering of music based on string compression*. In *Computer Music Journal*, volume 28(4):49–67, 2004.

[3] P. Grünwald and P. Vitányi: *Kolmogorov Complexity and Information Theory. With an Interpretation in Terms of Questions and Answers*. In *Journal of Logic, Language and Information*, volume 12(4):497–529, 2003.

[4] C. E. Shannon: *A Mathematical Theory of Communication*. In *The Bell System Technical Journal*, volume 27:379–423,623–656, 1948.

[5] D. H. Wolpert and W. G. Macready: *No free lunch theorems for optimization*. In *Evolutionary Computation, IEEE Transactions on*, volume 1(1):67–82, 1997.

[6] C. H. Bennett, P. Gács, M. Li, P. M. Vitányi, and W. H. Zurek: *Information distance*. In *Information Theory, IEEE Transactions on*, volume 44(4):1407–1423, 1998.

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

[8] M. Li, J. H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang: *An information-based sequence distance and its application to whole mitochondrial genome phylogeny*. In *Bioinformatics*, volume 17(2):149–154, 2001.

[9] Z. Dawy, J. Hagenauer, P. Hanus, and J. C. Mueller: *Mutual information based distance measures for classification and content recognition with applications to genetics.* In *ICC*, pages 820–824. 2005.

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

[11] R. Sibson: *SLINK: An optimally efficient algorithm for the single-link cluster method*. In *The Computer Journal*, volume 16(1):30–34, 1973.

[12] M. Weiser: *Programmers Use Slices when Debugging*. In *Commun. ACM*, volume 25(7):446–452, 1982.

[13] S. Chen and J. Reif: *Efficient lossless compression of trees and graphs*. In *Data Compression Conference, 1996. DCC '96. Proceedings*, pages 428–. 1996.