Algorithmic Clustering of Music[*]

Rudi Cilibrasi Paul Vitányi Ronald de Wolf
CWI
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
{cilibrar, paulv, rdewolf}@cwi.nl

Abstract

We present a method for hierarchical music clustering, based on compression of strings that represent the music pieces. The method uses no background knowledge about music whatsoever: it is completely general and can, without change, be used in different areas like linguistic classification, literature, and genomics. Indeed, it can be used to simultaneously cluster objects from completely different domains, like with like. It is based on an ideal theory of the information content in individual objects (Kolmogorov complexity), information distance, and a universal similarity metric. The approximation to the universal similarity metric obtained using standard data compressors is called "normalized compression distance (NCD)." Experiments using our CompLearn software tool show that the method distinguishes between various musical genres and can even cluster pieces by composer.

1. Introduction

The amount of digitized music available on the internet has grown dramatically in recent years, both in the public domain and on commercial sites. Napster and its clones are prime examples. Websites offering musical content in some form or other (MP3, MIDI, ...) need a way to organize their wealth of material; they need to somehow classify their files according to musical genres and subgenres, putting similar pieces together. The purpose of such organization is to enable users to navigate to pieces of music they already know and like, but also to give them advice and recommendations ("If you like this, you might also like..."). Currently, such organization is mostly done manually by humans, but some recent research has been looking into the possibilities of automating music classification.

[*] Supported in part by NWO, the NoE QUIPROCONE +IST–1999–29064, the ESF QiT Programmme, and the EU Fourth Framework BRA NeuroCOLT II Working Group EP 27150.

A human expert, comparing different pieces of music with the aim to cluster likes together, will generally look for certain specific similarities. Previous attempts to automate this process do the same. Generally speaking, they take a file containing a piece of music and extract from it various specific numerical features, related to pitch, rhythm, harmony etc. One can extract these using for instance Fourier transforms [26] or wavelet transforms [15]. The feature vectors corresponding to the various files are then classified or clustered using existing classification software, based on various standard statistical pattern recognition classifiers [26], Bayesian classifiers [12], hidden Markov models [7], ensembles of nearest-neighbor classifiers [15] or neural networks [12, 24]. For example, one feature would be to look for rhythm in the sense of beats per minute. One can make a histogram where each histogram bin corresponds to a particular tempo in beats-per-minute and the associated peak shows how frequent and strong that particular periodicity was over the entire piece. In [26] we see a gradual change from a few high peaks to many low and spread-out ones going from hip-hip, rock, jazz, to classical. One can use this similarity type to try to cluster pieces in these categories. However, such a method requires specific and detailed knowledge of the problem area, since one needs to know what features to look for.

Our aim is much more general. We do not look for similarity in specific features known to be relevant for classifying music; instead we apply a general mathematical theory of similarity. The aim is to capture, in a single similarity metric, *every effective metric*: effective versions of Hamming distance, Euclidean distance, edit distances [23], Lempel-Ziv distance [11], and so on. This metric should be so general that it works in every domain: music, text, literature, programs, genomes, executables, natural language determination, equally and simultaneously. Such a metric would be able to simultaneously detect *all* similarities between pieces that other effective metrics can detect. Rather surprisingly, such a "universal" metric indeed exists. It was developed in [17, 18, 19], based on the "information distance" of [20, 3]. Roughly speaking, two objects are deemed close if we can significantly "compress" one given the information in the other, the idea being that if two pieces are more similar, then we can more succinctly describe one given the other. The underlying mathematical theory is provably universal and is based on the ideal notion of Kolmogorov complexity, which unfortunately is not effectively computable. We replace the ideal but noncomputable Kolmogorov-based version by standard compression techniques. Theoretical analysis of the application of real-world compressors is given in [9]. (In contrast, a later and partially independent approach of [1, 2] for building language-trees—while citing [20, 3]—is by *ad hoc* arguments about empirical Shannon entropy and Kullback-Leibler distance resulting in non-metric distances.) The resulting distance is called "normalized compression distance (NCD)", and resulted in an open source software package [10] freely available on the web. The NCD metric appears to be truly universal in practice: it works well on all concrete examples we tried in very different application fields—the first completely automatic construction of the phylogeny tree based on whole mitochondrial genomes, [17, 18, 19], a completely automatic construction of a language tree for over 50 Euro-Asian languages [19], detecting plagiarism in student programming assignments [25], and phylogeny of chain letters [4], literature, astronomy, OCR [9].

In this paper we apply this compression-based method to the hierarchical clustering of pieces of music. We use hierarchical clustering because visual representation of the natural-data distance matrix using multidimensional scaling gives higher distortion of the distances involved, and is less informative, than hierarchical clustering, see [9]. We perform various experiments on sets of mostly classical pieces given as MIDI (Musical Instrument Digital Interface) files. This contrasts with most earlier research, where the music was digitized in some wave format or other, and often received in *mp3* or other compressed format. We compute the distances between all pairs of pieces, and then build a tree containing those pieces in a way that is consistent with those distances. First, as proof of principle, we run the program on three artificially generated data sets, where we know what the final answer should be. The program indeed classifies these perfectly. Secondly, we show that our program can distinguish between various musical genres (classical, jazz, rock) quite well. Thirdly, we experiment with various sets of classical pieces. The results are good (in the sense of conforming to our expectations) for small sets of data, but tend to be worse for large sets. This is an unavoidable consequence of the fact that while n objects with distances can be faithfully represented in n-dimensional space, representing the relations in 2-dimensional space (multidimensional clustering), or hierarchical (ternary) tree clustering, induces unavoidable distortion (like the Mercator projection of the earth sphere to a two-dimensional map). Increasing the number of objects also increases the number of distance requirements that cannot be satisfied, and hence the distortion. The method has received considerable media attention, e.g. [22, 27].

Related work: After a first version of our paper appeared on a preprint server [8], we learned of recent independent experiments on MIDI files [21]. Here, the matrix of distances is computed using the alternative compression-based approach of [1, 2] and the files are clustered on a Kohonen map rather than a tree. Their first experiment takes 17 classical piano pieces as input, and gives a clustering of comparable quality to ours. Their second experiment is on a set of 48 short artificial musical pieces (stimuli), and clusters these reasonably well in 8 categories.

Another very interesting line of music research using compression-based techniques may be found in the survey [13] and the references therein. Here the aim is not to cluster similar musical pieces together, but to model the *musical style* of a given MIDI file. For instance, given (part of) a piece by Bach, one would like to predict how the piece continues, or to algorithmically generate new pieces of music in the same style. Techniques based on Lempel-Ziv compression do a surprisingly good job at this.

A third related line of work is the area of "query by humming", for which see [14] and many later papers. Here a user "hums" a tune, and a program is supposed to find the piece of music (in some database) that is closest to the hummed tune. Clearly, any such approach will involve some quantitative measure of similarity. However, we are not aware of any compression-based similarity measure being used in this area.

2. Algorithmic Clustering

2.1. Kolmogorov complexity

Each object (in the application of this paper: each piece of music) is coded as a string x over a finite alphabet, say the binary alphabet. The integer $K(x)$ gives the length of the shortest compressed binary version from which x can be fully reproduced, also known as the *Kolmogorov complexity* of x. "Shortest" means the minimum taken over every possible decompression program, the ones that are currently known as well as the ones that are possible but currently unknown. We explicitly write only "decompression" because we do not even require that there is also a program that compresses the original file to this compressed version—if there is such a program then so much the better.

So $K(x)$ gives the length of the ultimate compressed version, say x^*, of x. This can be considered as the amount of information, number of bits, contained in the string. Similarly, $K(x|y)$ is the minimal number of bits (which we may think of as constituting a computer program) required to reconstruct x from y. In a way $K(x)$ expresses the individual "entropy" of x—the minimal number of bits to communicate x when sender and receiver have no knowledge where x comes from. For example, to communicate Mozart's "Zauberflöte" from a library of a million items requires at most 20 bits ($2^{20} \approx 1,000,000$), but to communicate it from scratch requires megabits. For more details on this pristine notion of individual information content we refer to [20].

2.2. Distance-based classification

As mentioned, our approach is based on a new very general similarity distance, classifying the objects in clusters of objects that are close together according to this distance. In mathematics, lots of different distances arise in all sorts of contexts, and one usually requires these to be a 'metric', since otherwise undesirable effects may occur. A metric is a distance function $D(\cdot,\cdot)$ that assigns a non-negative distance $D(a,b)$ to any two objects a and b, in such a way that $D(a,b) = 0$ only where $a = b$, $D(a,b) = D(b,a)$ (symmetry), and $D(a,b) \leq D(a,c) + D(c,b)$ (triangle inequality). We are interested in "similarity metrics". For example, if the objects are classical music pieces then the function $D(a,b) = 0$ if a and b are by the same composer and $D(a,b) = 1$ otherwise, is a similarity metric, albeit a somewhat elusive one. This captures only one, but quite a significant, similarity aspect between music pieces.

In [19], a new theoretical approach to a wide class of similarity metrics was proposed: their "normalized information distance" is a metric, and is universal in the sense that this single metric uncovers all similarities simultaneously that the metrics in the class uncover separately. This should be understood in the sense that if two pieces of music are similar (that is, close) according to the particular feature described by a particular metric, then they are also similar (that is, close) in the sense of the normalized information distance metric. This justifies calling the latter *the* similarity metric. Oblivious to the problem area concerned, simply using the distances according to the similarity metric, our method fully automatically classifies the objects concerned, be they music pieces, text corpora, or genomic data.

More precisely, the approach is as follows. Each pair of such strings x and y is assigned a distance

$$d(x,y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}. \qquad (1)$$

There is a natural interpretation to $d(x,y)$: If, say, $K(y) \geq K(x)$ then we can rewrite

$$d(x,y) = \frac{K(y) - I(x:y)}{K(y)},$$

where $I(x:y)$ is the information in y about x satisfying the symmetry property $I(x:y) = I(y:x)$ up to a logarithmic additive error [20]. That is, the distance $d(x,y)$ between x and y is the number of bits of information that is not shared between the two strings per bit of information that could be maximally shared between the two strings.

It is clear that $d(x,y)$ is symmetric, and in [19] it is shown that it is indeed a metric. Moreover, it is universal in the sense that every metric expressing some similarity that can be computed from the objects concerned is comprised (in the sense of minorized) by $d(x,y)$. It is these distances that we will use, albeit in the form of a rough approximation: for $K(x)$ we simply use standard compression software like 'gzip', 'bzip2', or 'PPMZ'. To compute the conditional version, $K(x|y)$ we use a sophisticated theorem, known as "symmetry of algorithmic information" in [20]. This says

$$K(y|x) \approx K(xy) - K(x), \qquad (2)$$

so to compute the conditional complexity $K(x|y)$ we can just take the difference of the unconditional complexities $K(xy)$ and $K(y)$. The theory based on real-world compressors is developed in [9], for a new theoretical class of compressors called "normal." For our normal real-world compressors (like gzip, bzip2, PPMZ) the resulting variant of $d(x,y)$ is called the *normalized compression distance (NCD).* It is shown to be a metric, we don't need (2), and we have approximate universality. (The Kolmogorov complexity case represents the ultimate normal compressor.) The NCD can be computed using our freely available CompLearn Toolkit [10].

2.3. Our quartet method

The above approach allows us to compute the distance between any pair of objects (any two pieces of music). We now need to cluster the objects, so that objects that are similar according to our metric are placed close together. Multidimensional non-hierarchical clustering turns out to distort the clusters and give poor visual information. We chose hierarchical clustering in ternary trees (each internal node has three edges) to represent the hierarchical information in the distance matrix as well as possible [9]. We need a sensitive method to extract the information contained in the distance matrix. For example, our experiments showed that reconstructing a minimum spanning tree is not sensitive enough and gives poor results. The "quartet method" in contrast is quite sensitive. The idea is as follows: we consider every group of four elements from our set of n elements (in this case, musical pieces); there are $\binom{n}{4}$ such groups. From each group u,v,w,x we construct a tree of arity 3, which implies that the tree consists of two subtrees of two leaves each. Let us call such a tree a *quartet*. There are three possibilities denoted (i) $uv|wx$, (ii) $uw|vx$, and (iii) $ux|vw$, where a vertical bar divides the two pairs of leaf nodes into two disjoint subtrees.

For any given tree T and any group of four leaf labels u, v, w, x, we say T is *consistent* with $uv|wx$ if and only if the path from u to v does not cross the path from w to x. Note that exactly one of the three possible quartets for any set of 4 labels must be consistent for any given tree. We may think of a large tree having many smaller quartet trees embedded within its structure. We formulate a possibly novel, sensitive, cost optimization problem: The cost of a quartet is defined as the sum of the distances between each pair of neighbors; that is, $C_{uv|wx} = d(u,v) + d(w,x)$. The total cost C_T of a tree T with a set N of leaves (external nodes of degree 1) is defined as $C_T = \sum_{\{u,v,w,x\} \subseteq N} \{C_{uv|wx} : T$ is consistent with $uv|wx\}$—the sum of the costs of all its consistent quartets. First, we generate a list of all possible quartets for all four-tuples of labels under consideration. For each group of three possible quartets for a given set of four labels u, v, w, x, calculate a best (minimal) cost $m(u,v,w,x) = \min\{C_{uv|wx}, C_{uw|vx}, C_{ux|vw}\}$, and a worst (maximal) cost $M(u,v,w,x) = \max\{C_{uv|wx}, C_{uw|vx}, C_{ux|vw}\}$. Summing all best quartets yields the best (minimal) cost $m = \sum_{\{u,v,w,x\} \subseteq N} m(u,v,w,x)$. Conversely, summing all worst quartets yields the worst (maximal) cost $M = \sum_{\{u,v,w,x\} \subseteq N} M(u,v,w,x)$. For some distance matrices, these minimal and maximal values can not be attained by actual trees; however, the score C_T of every tree T will lie between these two values. In order to be able to compare tree scores in a more uniform way, we now rescale the score linearly such that the worst score maps to 0, and the best score maps to 1, and term this the *normalized tree benefit score* $S(T) = (M - C_T)/(M - m)$. Our goal is to find a full tree with a maximum value of $S(T)$, which is to say, the lowest total cost. This optimization problem reduces to problems that are known to be NP-hard [16], which means that it is infeasible in practice, but we can sometimes solve it and always approximate it. Adapting current methods in [5] results in far too computationally intensive calculations; they run many months or years on moderate-sized problems of 30 objects. We have designed a simple heuristic method for our problem based on randomization and hill-climbing. First, a random tree with $2n-2$ nodes is created, consisting of n leaf nodes (with 1 connecting edge) labeled with the names of musical pieces, and $n-2$ non-leaf or *internal* nodes. Each internal node has exactly three connecting edges. For this tree T, we calculate the total cost of all consistent quartets, and invert and scale this value to find $S(T)$. Typically, a random tree will be consistent with around $\frac{1}{3}$ of all quartets. Now, this tree is denoted the currently best known tree, and is used as the basis for further searching. We define a simple mutation on a tree as one of the three possible transformations:

1. A *leaf swap*, which consists of randomly choosing two leaf nodes and swapping them.
2. A *subtree swap*, which consists of randomly choosing two internal nodes and swapping the subtrees rooted at those nodes.
3. A *subtree transfer*, whereby a randomly chosen subtree (possibly a leaf) is detached and reattached in another place, maintaining arity invariants.

Each of these simple mutations keeps invariant the number of leaf and internal nodes in the tree; only the structure and placements change. Define a full mutation as a sequence of at least one but potentially many simple mutations, picked according to the following distribution. First we pick the number k of simple mutations that we will perform with probability 2^{-k}. For each such simple mutation, we choose one of the three types listed above with equal probability. Finally, for each of these simple mutations, we pick leaves or internal nodes, as necessary. Notice that trees which are close to the original tree (in terms of number of simple mutation steps in between) are examined often, while trees that are far away from the original tree will eventually be examined, but not very frequently. So in order to search for a better tree, we simply apply a full mutation on T to arrive at T', and then calculate $S(T')$. If $S(T') > S(T)$, then keep T' as the new best tree. Otherwise, try a new different tree and repeat. If $S(T')$ ever reaches 1, then halt, outputting the best tree. Otherwise, run until it seems no better trees are being found in a reasonable amount of time, in which case the approximation is complete.

Note that if a tree is ever found such that $S(T) = 1$, then we can stop because we can be certain that this tree is optimal, as no tree could have a lower cost. In fact, this perfect tree result is achieved in our artificial tree reconstruction experiment (Section 4.1) reliably in less than ten minutes. For real-world data, $S(T)$ reaches a maximum somewhat less than 1, presumably reflecting inconsistency in the distance matrix data fed as input to the algorithm, or indicating a search space too large to solve exactly. On many typical problems of up to 40 objects this tree-search gives a tree with $S(T) \geq 0.9$ within half an hour. Progress occurs typically in a sigmoidal fashion towards a maximal value ≤ 1. For large numbers of objects, tree scoring itself can be slow (as this takes order n^4 computation steps), and the space of trees is also large, so the algorithm may slow down substantially. For larger experiments, we use a C++/Ruby implementation with MPI (Message Passing Interface, a common standard used on massively parallel computers) on a cluster of workstations in parallel to find trees more rapidly.

3. Details of our implementation

The software that we developed for the experiments of this paper (and for later experiments reported in [9]) is freely available [10]. We downloaded 118 separate MIDI files selected from a range of classical composers, as well as some popular music. We then preprocessed these MIDI files to make them more uniform. This is done to keep the experiments "honest": We want to analyze just the musical component, not the title indicator in the MIDI file, nor the sequencer's name, or author/composer's name, nor sequencing program used, nor any of the many other non-musical data that can be incorporated in the MIDI file. We strip off this information from the MIDI file to avoid detecting similarity between files for non-musical reasons, for example, like being prepared by the same source.

The preprocessor extracts just MIDI Note-On and Note-Off events. These events were then converted to a player-piano style representation, with time quantized in 0.05 second intervals. All instrument indicators, MIDI Control signals, and tempo variations were ignored. For each track in the MIDI file, we calculate two quantities: an *average volume* and a *modal note* ("modal" is used here in a statistical sense, not in a musical sense). The average volume is calculated by averaging the volume (MIDI Note velocity) of all notes in the track. The modal note is defined to be the note pitch that sounds most often in that track. If this is not unique, then the lowest such note is chosen. The modal note is used as a key-invariant reference point from which to represent all notes. It is denoted by 0, higher notes are denoted by positive numbers, and lower notes are denoted by negative numbers. A value of 1 indicates a half-step above the modal note, and a value of -2 indicates a whole-step below the modal note. The modal note is written as the first byte of each track. For each track, we iterate through each 0.05-sec time sample in order, outputting a single signed 8-bit value for each currently sounding note (ordered from lowest to highest). Two special values are reserved to represent the end of a time step and the end of a track. The tracks are sorted according to decreasing average volume, and then output in succession. The preprocessing phase does not significantly alter the musical content of the MIDI file: the preprocessed file sounds almost the same as the original.

These preprocessed MIDI files are then used as input to the compression stage for distance matrix calculation and subsequent tree search. We chose to use the compression program 'bzip2' for our experiments. Unlike the well known dictionary-based Lempel-Zip compressors, bzip2 transforms a file into a data-dependent permutation of itself by using the Burrows Wheeler Transform [6]. Very briefly: The BWT operates on the file as well as on all of its rotations; the algorithm sorts this block of rotations and uses a move-to-front encoding scheme to focus the redundancy in the file into simple statistical biases that can be used by an entropy coder in the output stage without context.

The resulting matrix of pairwise distances is then fed into our tree construction program, described in detail in the previous section, which lays out the experiment's MIDI files in tree format. Everything runs on 1.5GHz pentiums with an insignificant memory footprint.

4. Results

4.1. Three controlled experiments

With the natural data sets of music pieces that we use, one may have the preconception (or prejudice) that music by Bach should be clustered together, music by Chopin should be clustered together, and so should music by rock stars. However, the preprocessed music files of a piece by Bach and a piece by Chopin, or the Beatles, may resemble one another more than two different pieces by Bach—by accident or indeed by design and copying. Thus, natural data sets may have ambiguous, conflicting, or counterintuitive outcomes. In other words, the experiments on actual pieces have the drawback of not having one clear "correct" answer that can function as a benchmark for assessing our experimental outcomes. Before describing the experiments we did with MIDI files of actual music, we discuss three experiments that show that our program indeed does what it is supposed to do—at least in artificial situations where we know in advance what the correct answer is. The similarity machine consists of two parts: (i) extracting a distance matrix from the data, and (ii) constructing a tree from the distance matrix using our novel quartet-based heuristic.

Testing the quartet-based tree construction: We first test whether the quartet-based tree construction heuristic is trustworthy: We generated a random ternary tree T with 18 leaves, and derived a distance metric from it by defining the distance between two nodes as follows: Given the length of the path from a to b, in an integer number of edges, as $L(a,b)$, let

$$d(a,b) = \frac{L(a,b)+1}{18},$$

except when $a = b$, in which case $d(a,b) = 0$. It is easy to verify that this simple formula always gives a number between 0 and 1, and is monotonic with path length. Given only the 18×18 matrix of these normalized distances, our quartet method exactly reconstructed the original tree with $S(T) = 1$.

Testing the similarity machine on artificial data: Given that the tree reconstruction method is accurate on clean consistent data, we tried whether the full procedure works in an acceptable manner when we know what the outcome should be like. For reasons of space we omit the details and resulting tree, but note that it had clustering occur exactly as we would expect. The $S(T)$ score is 0.905.

Testing the similarity machine on natural data: We test gross classification of files based on markedly different file types. Here, we chose several files: (i) Four mitochondrial gene sequences, from a black bear, polar bear, fox, and rat; (ii) Four excerpts from the novel *The Zeppelin's Passenger* by E. Phillips Oppenheim; (iii) Four MIDI files without further processing; two from Jimi Hendrix and two movements from Debussy's Suite bergamasque; (iv) Two Linux x86 ELF executables (the *cp* and *rm* commands); and (v) Two compiled Java class files. As expected, the program correctly classifies each of the different types of files together with like near like. The result is reported in Figure 1 with $S(T)$ equal to 0.984. This experiment shows the power and universality of the method: no features of any specific domain of application is used.

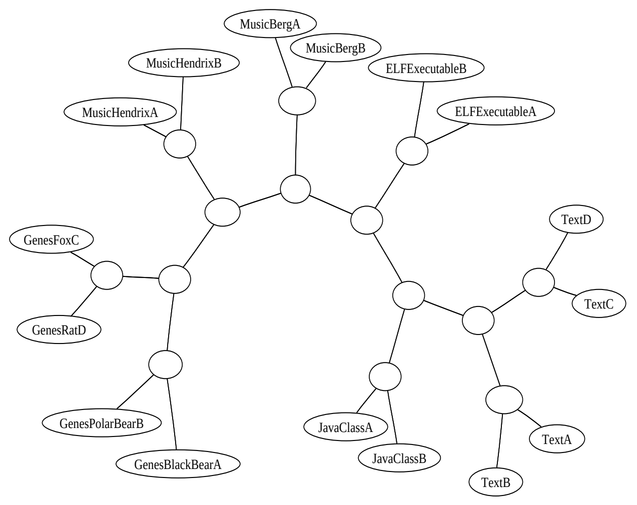

Figure 1. Classification of different file types

4.2. Music genre classification

The limited available space doesn't allow many pictures, so we briefly discuss the results. For the full paper, see [8]. Before testing whether our program can see the distinctions between various classical composers, we first show that it can distinguish between three broader musical genres: classical music, rock, and jazz. This should be easier than making distinctions "within" classical music. For the genre-experiment we used 12 classical pieces from Bach, Chopin, and Debussy, 12 jazz pieces from Miles Davis, John Coltrane and the like, and 12 rock pieces from The Beatles, The Police, etc. The output tree (Figure 2) has $S(T)$ score 0.858. All musical pieces used are listed in the tables in the full paper.

The discrimination between the 3 genres is good but not perfect. The upper-right branch of the tree contains 10 of the 12 jazz pieces, but also Chopin's Prélude no. 15 and a Bach Prelude. The two other jazz pieces, Miles Davis's "So what" and John Coltrane's "Giant steps" are placed elsewhere in the tree, perhaps according to some kinship that now escapes us but can be identified by closer studying of the objects concerned. Of the rock pieces, 9 are placed close together in the lower-left branch, while Hendrix's "Voodoo chile", Rush's "Yyz", and Dire Straits' "Money for nothing" are further away. In the case of the Hendrix piece this may be explained by the fact that it hovers between the jazz and rock genres. Most of the classical pieces are in the lower-right and middle part of the tree. Surprisingly, 2 of the 4 Bach pieces are placed elsewhere. It is not clear why this happens and may be considered an error of our program, since we perceive the 4 Bach pieces to be very close, both structurally and melodically. However, Bach's is a seminal music and has been copied and cannibalized in all kinds of recognizable or hidden manners; closer scrutiny could reveal likenesses in its present company that are not now apparent to us. In effect our similarity engine aims at the ideal of a perfect data mining process, discovering unknown features in which the data can be similar.

4.3. Classical piano music

We then tested our method on three sets, of increasing size, of classical piano music. The smallest set encompasses the 4 movements from Debussy's Suite bergamasque, 4 movements of book 2 of Bach's Wohltemperierte Klavier, and 4 preludes from Chopin's opus 28. As one can see in Figure 3, our program does a pretty good job at clustering these pieces. The $S(T)$ score is also high: 0.958. The 4 Debussy movements form one cluster, as do the 4 Bach pieces. The only imperfection in the tree, judged by what one would intuitively expect, is that Chopin's Prélude no. 15 lies a bit closer to Bach than to the other 3 Chopin pieces. This Prélude no 15, in fact, consistently forms an odd-one-out in our other experiments as well. There is some musical truth to this, as no. 15 may be perceived as the most eccentric among the 24 Préludes of Chopin's opus 28.

We further tested the method with a medium-sized set that added 20 pieces to the small set, which gave an $S(T)$ score slightly lower than in the small set experiment: 0.895; a large set of 60 pieces where the $S(T)$ score dropped further from that of the medium-sized set to 0.844; more complicated music, namely 34 symphonic pieces, which resulted in an $S(T)$ score of 0.860. In all cases the $S(T)$ score is reliable with respect to what our intuition tells us. Note that a lower $S(T)$ score only indicates that the corresponding matrix of distances is not faithfully represented by the tree. Whether the distance matrix itself satisfies our preconceived

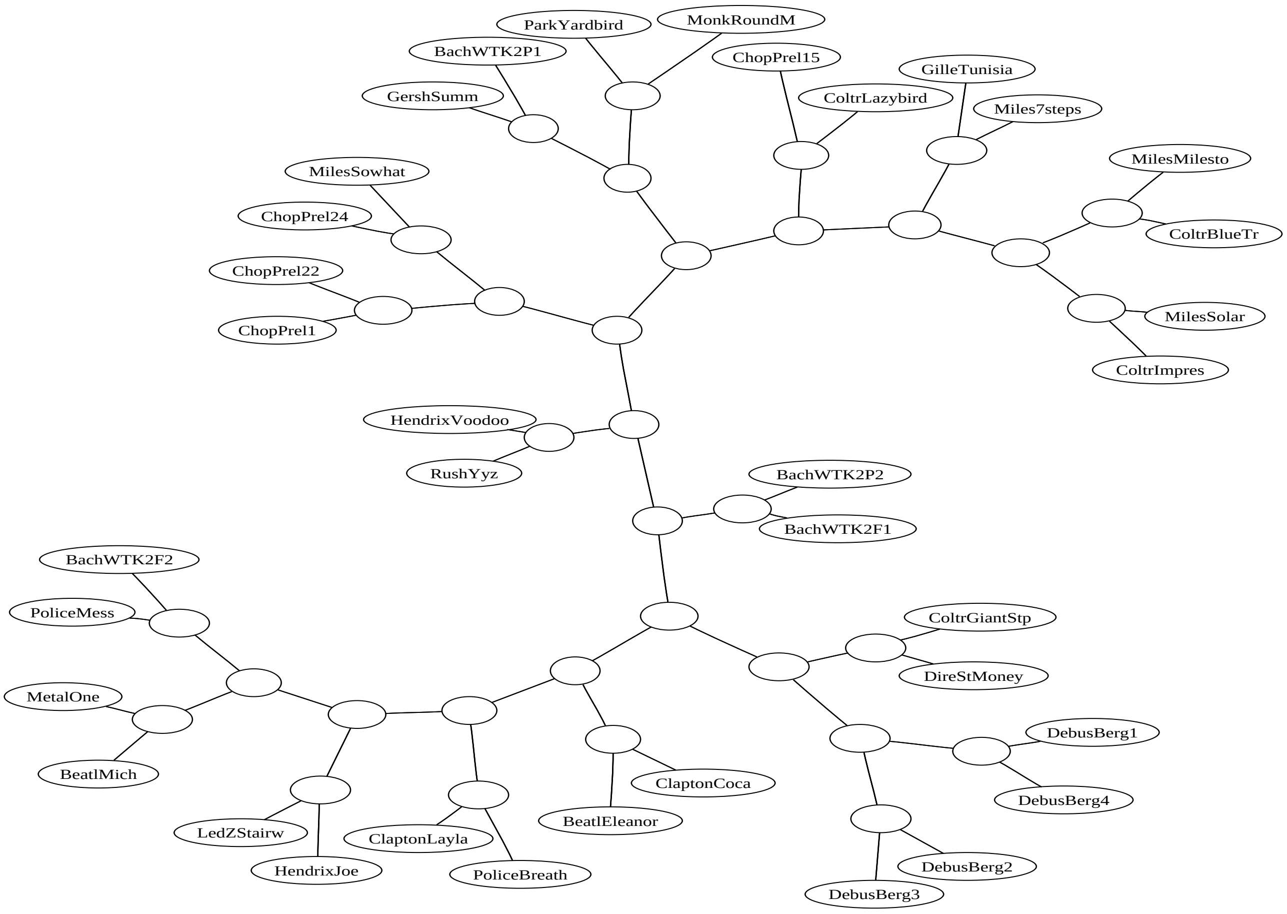

Figure 2. Output for the 36 pieces from 3 genres

ideas about musical similarity is a separate issue, which we do not address here.

5. Summary and conclusion

In this paper we reported on experiments that cluster sets of MIDI files by means of compression. The intuitive idea is that two files are closer to the extent that one can be compressed better "given" the other. Thus the notion of compression induces a *similarity metric* on strings in general and MIDI files in particular. Our method derives from the notion of Kolmogorov complexity, which describes the ultimate limits of compression. As a theoretical approach this is provably universal and optimal. The actual implementation, however, is by necessity non-optimal because the uncomputable Kolmogorov complexity has to be replaced by some practical compressor (we used bzip2 here, though others give similar results). We described various experiments where we first computed the matrix of pairwise distances between the various MIDI files involved, and then used a new heuristic tree construction algorithm to lay out the pieces in a tree, in accordance with the computed distances. We want to stress again that our method does not rely on any music-theoretical knowledge or analysis, but only on general-purpose compression techniques. The versatility and general-purpose nature of our method is also exemplified by the range of later experiments reported in the subsequent paper [9].

References

[1] D. Benedetto, E. Caglioti, and V. Loreto. Language trees and zipping. *Physical Review Letters*, 88:4, 048702, 2002.

[2] Ph. Ball. Algorithm makes tongue tree. *Nature*, January 22, 2002.

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

[4] C.H. Bennett, M. Li, B. Ma. Chain letters and evolutionary histories. *Scientific American*, 76–81, June 2003.

[5] D. Bryant, V. Berry, P. Kearney, M. Li, T. Jiang, T. Wareham and H. Zhang. A practical algorithm for recovering the best

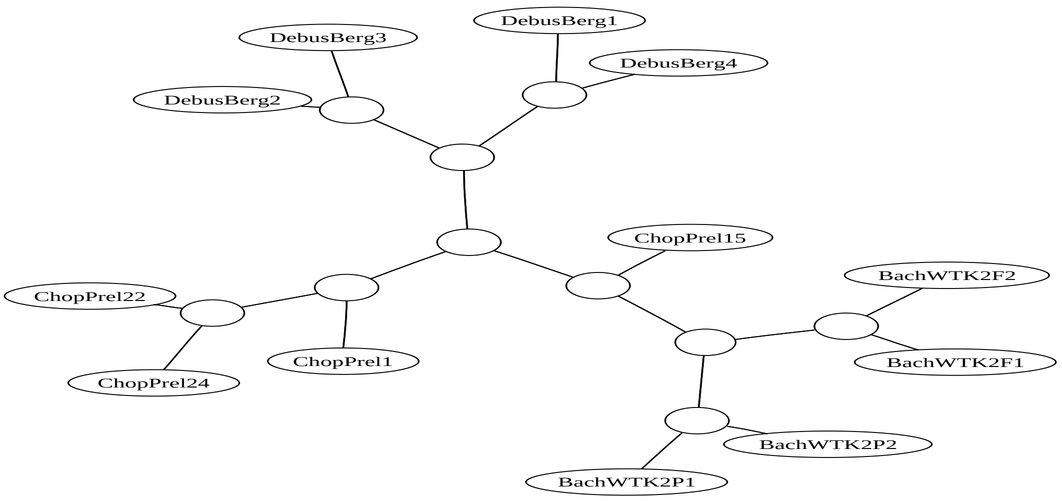

Figure 3. Output for the 12-piece set

supported edges of an evolutionary tree. *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, 287–296, 2000.

[6] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. Technical report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[7] W. Chai and B. Vercoe. Folk music classification using hidden Markov models. *Proc. of International Conference on Artificial Intelligence*, 2001.

[8] R. Cilibrasi, P. Vitányi, and R. de Wolf. Algorithmic clustering of music. http://arxiv.org/abs/cs.SD/0303025. Different and extended version accepted for publication in *Computer Music Journal*.

[9] R. Cilibrasi and P. Vitányi. Clustering by compression. http://arxiv.org/abs/cs.CV/0312044

[10] CompLearn Toolkit :: Machine Learning Via Compression, written by R. Cilibrasi, http://complearn.sourceforge.net/

[11] G. Cormode, M. Paterson, S. Sahinalp, and U. Vishkin. Communication complexity of document exchange. *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, pp. 197–206, 2000.

[12] R. Dannenberg, B. Thom, and D. Watson. A machine learning approach to musical style recognition. *Proc. International Computer Music Conference*, pp. 344–347, 1997.

[13] S. Dubnov, G. Assayag, O. Lartillot, and G. Bejerano. Using machine-learning methods for musical style modeling. *Computer 36*(10):73–80, 2003. IEEE.

[14] A. Ghias, J. Logan, D. Chamberlin, and B.C. Smith. Query by humming: Musical information retrieval in an audio database. *Proc. of ACM Multimedia Conference*, pp. 231–236, 1995.

[15] M. Grimaldi, A. Kokaram, and P. Cunningham. Classifying music by genre using the wavelet packet transform and a round-robin ensemble. Technical report TCD-CS-2002-64, Trinity College Dublin, 2002. http://www.cs.tcd.ie/publications/tech-reports/reports.02/TCD-CS-2002-64.pdf

[16] T. Jiang, P. Kearney, and M. Li. A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. *SIAM J. Computing*, 30(6):1942–1961, 2001.

[17] 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. *Bioinformatics*, 17(2):149–154, 2001.

[18] M. Li and P.M.B. Vitányi. Algorithmic complexity. In *International Encyclopedia of the Social & Behavioral Sciences*, pp. 376–382, N.J. Smelser and P.B. Baltes, Eds., Pergamon, Oxford, 2001/2002.

[19] M. Li, X. Chen, X. Li, B. Ma, P. Vitányi. The similarity metric. *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms*, pp. 863–872, 2003.

[20] M. Li and P.M.B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, New York, 2nd Edition, 1997.

[21] A. Londei, V. Loreto, and M. O. Belardinelli. Musical style and authorship categorization by informative compressors. *Proc. 5th Triennial ESCOM Conference*, pp. 200–203, 2003.

[22] H. Muir. Software to unzip identity of unknown composers. *New Scientist*. April 12, 2003.

[23] K. Orpen and D. Huron. Measurement of similarity in music: A quantitative approach for non-parametric representations. *Computers in Music Research 4*, 1–44, 1992.

[24] P. Scott. Music classification using neural networks, 2001. http://www.stanford.edu/class/ee373a/musicclassification.pdf

[25] Shared Information Distance or Software Integrity Detection, Computer Science, University of California, Santa Barbara, http://dna.cs.ucsb.edu/SID/

[26] G. Tzanetakis and P. Cook. Music genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002.

[27] K. Patch. Software sorts tunes. *Technology Research News*, April 23/30, 2003.