Algorithmic Clustering of Music Based on String Compression

Rudi Cilibrasi,* Paul Vitányi,*† and Ronald de Wolf*
*Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam, The Netherlands
†Institute for Logic, Language, and Computation
University of Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam, The Netherlands
{Rudi.Cilibrasi, Paul.Vitanyi,
Ronald.de.Wolf}@cwi.nl

All musical pieces are similar, but some are more similar than others. Apart from serving as an infinite source of discussion (''Haydn is just like Mozart—No, he's not!''), such similarities are also crucial for the design of efficient music information retrieval systems. 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. Web sites offering musical content in some form like MP3, MIDI, or other, 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, or based on patterns in the purchasing behaviors of customers. However, some recent research has been examining the possibilities of automating music classification.

A human expert, comparing different pieces of music with the goal of clustering similar works 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 such features using, for instance, Fourier transforms (Tzanetakis and Cook 2002) or wavelet transforms (Grimaldi, Kokaram, and Cunningham 2002). 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 (Tzanetakis and Cook 2002), Bayesian classifiers (Dannenberg, Thom, and Watson 1997), hidden Markov models (Chai and Vercoe 2001), ensembles of nearest-neighbor classifiers (Grimaldi, Kokaram, and Cunningham 2002), or neural networks (Dannenberg, Thom, and Watson 1997; see also a paper by Paul Scott available online at www.stanford.edu/class/ee373a/musicclassification.pdf).

For example, one feature to look for would be tempo in the sense of beats per minute. One can make a histogram in which 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 Tzanetakis and Cook (2002), we see a gradual change from a few high peaks to many low and spread-out ones proceeding from hip-hop, rock, and 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, because one needs to know the features for which to look.

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 (Orpen and Huron 1992), Lempel-Ziv distance (Cormode et al. 2000), and so on. 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 Li et al. (2001), Li

Computer Music Journal, 28:4, pp. 49–67, Winter 2004

and Vitányi (2002), and Li et al. (2003), based on the ''information distance'' of Li and Vitányi (1997) and Bennett et al. (1998). 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. Here compression is based on the ideal mathematical notion of Kolmogorov complexity, the length of the shortest compressed code from which the original object can be losslessly reproduced by an effective decompressor.

Unfortunately, this limit to what current and future compressors can do comes at the price of not being computable, but it is a powerful theoretical analysis tool we can use to advantage. It is well known that when a pure mathematical theory is applied to the real world, for example in hydrodynamics or in physics in general, we can only approximate the theoretical ideal. But still, the theory gives a framework and foundation for the applied science. In practice, we replace compression up to the Kolmogorov complexity by standard real-world compression techniques. We lose theoretical optimality in some cases, but we gain an efficiently computable similarity metric, the Normalized Compression Distance (NCD), intended to approximate the theoretical ideal. In contrast, a later and partially independent compression-based approach of Benedetto, Caglioti, and Loreto (2002) for building language-trees—although building on Li and Vitányi (1997) and Bennett et al. (1998)—is by essentially ad hoc arguments.

Earlier research has demonstrated that this new universal similarity metric works well on concrete examples in very different application fields, for example the first completely automatic construction of the phylogeny tree based on whole mitochondrial genomes (Li et al. 2001; Li and Vitányi 2002; Li et al. 2003), and a completely automatic construction of a language tree for over 50 Euro-Asian languages (Li et al. 2003). Other applications detected plagiarism in student computer-programming assignments (at the University of Santa Barbara in 2001; see dna.cs.ucsb.edu/SID) and phylogeny of chain letters. This gives evidence that, in practice, the approach mimics the ideal performance of Kolmogorov complexity, but it does not supply theoretical justification.

The method is implemented and available as public software, and is robust under choice of different compressors (Cilibrasi 2003). A subsequent article (available online at arxiv.org/abs/cs.CV/0312044) presents a theoretical analysis of the method based on real-world compressors and proves metricity and other required properties. To substantiate the claims of universality and robustness, evidence is reported there of successful application in areas as diverse as genomics, virology, languages, literature, handwritten digits, astronomy, and combinations of objects from completely different domains, using statistical, dictionary, and block-sorting compressors.

In this article, we apply this compression-based method to the classification of pieces of music. We perform various experiments on sets of pieces (mostly classical) given as MIDI files. We compute the distances between all pairs of pieces, resulting in a distance matrix of pairwise NCDs. To extract the maximal information in the distance matrix for visual display, we cluster the data hierarchically. This results in a tree containing those pieces in a way that is consistent with the computed distances. Because it is rather difficult to objectively judge the quality of the hierarchical clusters resulting from experiments on pieces of real music, we first tested our method on a number of artificially generated data sets where we know exactly what the resulting tree should be. Quite reassuringly, our method produced exactly the correct results in these cases. The details of these artificial experiments can be found online at arxiv.org/abs/cs.CV/0312044; for reasons of space, we report on only the actual musical experiments here.

First, we show that our program can distinguish between various musical genres (classical, jazz, and rock) quite well. Second, we experiment with various sets of classical piano pieces. Third, we cluster a large set of movements from symphonies by Mozart, Beethoven, and others. The results are quite good (in the sense of conforming well to our expectations) for small sets of data, but they tend to get a bit worse for large sets. Considering the fact that the method knows nothing about music, or, in-

deed, about any of the other areas we have applied it to elsewhere, one is reminded of Dr. Johnson's remark about a dog's walking on his hind legs: ''It is not done well; but you are surprised to find it done at all.''

The article is organized as follows. We first summarize related work, and then we give a brief domain-independent overview of compression-based clustering: the ideal distance metric based on Kolmogorov complexity and the quartet method that turns the matrix of distances into a tree. Next, we give the details of the current application to music, the specific file formats used, and so on. In the following section, we report the results of our experiments, and we conclude with some directions for future research.

Related Work

After a first version of our article appeared on a preprint server (arxiv.org/abs/cs.SD/0303025), we learned of recent independent experiments on MIDI files (Londei, Loreto, and Belardinelli 2003). In those experiements, the matrix of distances is computed using the alternative compression-based approach of Benedetto, Caglioti, and Loreto (2002) and Ball (2002), 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 into eight categories.

Another very interesting line of music research using compression-based techniques may be found in the survey by Dubnov et al. (2003) and the references therein. In that survey, the aim is not to cluster similar musical pieces together, but to model the musical style of a given MIDI file. For instance, given a portion 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,'' as in Ghias et al. (1995) and many later articles. 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.

Algorithmic Clustering

Kolmogorov Complexity

Each object—in the application of this article, 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—both those that are currently known as well as those 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.

Technically, the definition of Kolmogorov complexity is as follows. First, we fix a syntax for expressing all and only computations (computable functions). This can be in the form of an enumeration of all Turing machines, but also an enumeration of all syntactically correct programs in some universal programming language like Java, Lisp, or C. We then define the Kolmogorov complexity of a finite binary string as the length of the shortest Turing machine, Java program, etc., in our chosen syntax. Which syntax we take is unimportant, but we must adhere to our choice. This choice assigns a definite positive integer as the Kolmogorov complexity to each finite string.

Though defined in terms of a particular machine model, the Kolmogorov complexity is machine-independent up to an additive constant and acquires an asymptotically universal and absolute character through Church's thesis, which states that a universal Turing machine can simulate any

other effective computational process. The Kolmogorov complexity of an object can be viewed as an absolute and objective quantification of the amount of information in it. This leads to a theory of absolute-information contents of individual objects, in contrast to classic information theory, which deals with average information to communicate objects produced by a random source.

So $K(x)$ gives the length of the ultimate compressed version, of x. This can be considered as the amount of information—i.e., 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 the textbook by Li and Vitányi (1997).

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, different distances arise in all sorts of contexts, and one usually requires these to be a *metric*, because 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 the following properties hold:

1. $D(a,b)=0$, only where $a=b$;
2. $D(a,b)=D(b,a)$ (symmetry); and
3. $D(a,b) \leq D(a,c)+D(c,b)$ (triangle inequality).

A familiar example of a metric is the Euclidean metric, the everyday distance $e(a,b)$ between two objects a,b expressed in, say, meters. Clearly, this distance satisfies the properties $e(a,a)=0$, $e(a,b)=e(b,a)$, and $e(a,b) \leq e(a,c)+e(c,b)$. (Substitute $a=$ Amsterdam, $b=$ Brussels, and $c=$ Chicago.) We are interested in *similarity metrics*. For example, if the objects are classical music pieces, then an example similarity metric might be $D(a,b)=0$ if a and b are by the same composer, and $D(a,b)=1$ otherwise. This captures only one, but a quite significant, similarity aspect between pieces of music.

Li et al. (2003) propose a new theoretical approach to a wide class of similarity metrics: their "normalized information distance" is a metric. It is universal in the sense that this single metric uncovers all similarities simultaneously that the metrics in the class uncover separately, in a technical sense described below. This should be understood in the sense that if two pieces of music are similar (i.e., "close") according to the particular feature described by a particular metric, then they are also similar (i.e., "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, the method fully automatically classifies the objects concerned—be they musical pieces, text corpora, or genomic data.

More precisely, the approach is as follows. Each pair of 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

$$d(x,y)=\frac{K(y)-I(x:y)}{K(y)} \qquad (2)$$

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, and hence called the (algorithmic) *mutual information* (Li and Vitányi 1997). That is, the distance $d(x,y)$ between x and y is 1 minus the "normalized" mutual information between the two strings.

It is clear that $d(x,y)$ is symmetric, and in Li et al. (2003) it is shown that it is indeed a metric. Moreover, they show that d is universal in the sense that every metric expressing some similarity that can be computed from the objects concerned is

subsumed by $d(x,y)$: for every appropriately normalized metric f and every x and y, we have $d(x,y) \leq f(x,y) + O((\log k)/k)$, where $k = \max\{K(x),K(y)\}$. Informally speaking, this says that if x and y are close according to any well-behaved metric f, then x and y are also close according to the universal metric d. It is these distances $d(x,y)$ 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 from Li and Vitányi (1997). This says that

$$K(y|x) \approx K(xy) - K(x), \quad (3)$$

Here, xy denotes the concatenation of the strings x and y. Accordingly, to compute the conditional complexity $K(x|y)$, we can just take the difference of the unconditional complexities $K(xy)$ and $K(y)$. This allows us to approximate $d(x,y)$ for every pair x,y.

Caveats

Our actual practice falls short of the ideal theory in at least three respects. First, the claimed universality of the similarity distance $d(x,y)$ holds only for indefinitely long sequences x,y. Once we consider strings x,y of definite length n, the similarity distance is only universal with respect to ''simple'' computable normalized information distances, where ''simple'' means that they are computable by programs of length, say, logarithmic or polylogarithmic in n. This reflects the fact that, technically speaking, our universal distance can also be viewed as a weighted sum of all similarity distances from the class of distances considered with respect to the objects considered. Only similarity distances of which the complexity is small (which means that their weight is large) with respect to the size of the data concerned significantly contribute to this sum.

Second, the Kolmogorov complexity is not computable, and it is in principle impossible to compute how far off the Normalized Compression Distance, or NCD, is from the Kolmogorov metric. Thus, we cannot in general know how well we are doing using the NCD.

Third, to approximate the NCD, we use standard compression programs like gzip, bzip2, and PPMZ. Whereas better compression of a string will always approximate the Kolmogorov complexity better, this may not be true for the NCD. Owing to its arithmetic form—subtraction and division—it is theoretically possible that while all items in the formula get better compressed, the improvement is not the same for all items, and the NCD value moves away from the asymptotic value. In our experiments, we have not observed this behavior in a noticeable fashion. Formally, the quality of approximation is given in an article by two of the authors available online at arxiv.org/abs/cs.CV/0312044. A further problem is that the triangle inequality need no longer be satisfied when we replace Kolmogorov complexity by the outcome of a real-world compression program to obtain the NCD. In our online article, it is shown that under mild assumptions on the compressor, the NCD is a metric. Experiments have shown that many standard real-world compressors satisfy those assumptions.

The Quartet Method

Given a set of objects, the pairwise NCDs form the entries of a distance matrix representing the distances between all pairs of objects. This distance matrix contains the pairwise relations in raw form; however, in this format, that information is not easily usable. Just as the distance matrix is a reduced form of information representing the original data set, we now must reduce the information even further to achieve a cognitively acceptable format like data clusters. To extract a hierarchy of clusters from the distance matrix, we determine a dendrogram (e.g., binary tree) that agrees with the distance matrix according to a cost measure. This allows us to extract more information from the data than just flat clustering (determining disjoint clusters in dimensional representation).

Clusters are groups of objects that are similar according to our metric. There are various ways to cluster. Our aim is to analyze data sets for which the number of clusters is not known a priori and the data are not labeled. As stated in Duda, Hart,

and Stork (2001), conceptually simple, hierarchical clustering is among the best known unsupervised methods in this setting, and the most natural way is to represent the relations in the form of a dendrogram, which is customarily a directed binary tree or undirected ternary tree. To construct the tree from a distance matrix with entries consisting of the pairwise distances between objects, we use the quartet method. This is a matter of choice only; other methods may work equally well. With an increasing number of data items, the projection of the NCD matrix information into the tree representation format gets increasingly distorted. A similar situation arises in using alignment cost in genomic comparisons. Experience shows that in both cases, the hierarchical clustering methods seem to work best for small sets of data (up to 25 items), and it deteriorates for larger sets (say 40 items or more). A standard solution to hierarchically cluster larger sets of data is to cluster nonhierarchically first, by say multidimensional scaling of k-means, available in standard packages, for instance MATLAB, and then applying hierarchical clustering on the emerging clusters.

We use the *quartet method*. 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 where each internal node has 3 neighbors, 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 (1) $uv|wx$, (2) $uw|vx$, and (3) $ux|vw$, where a vertical bar divides the two pairs of leaf nodes into two disjoint subtrees (see Figure 1).

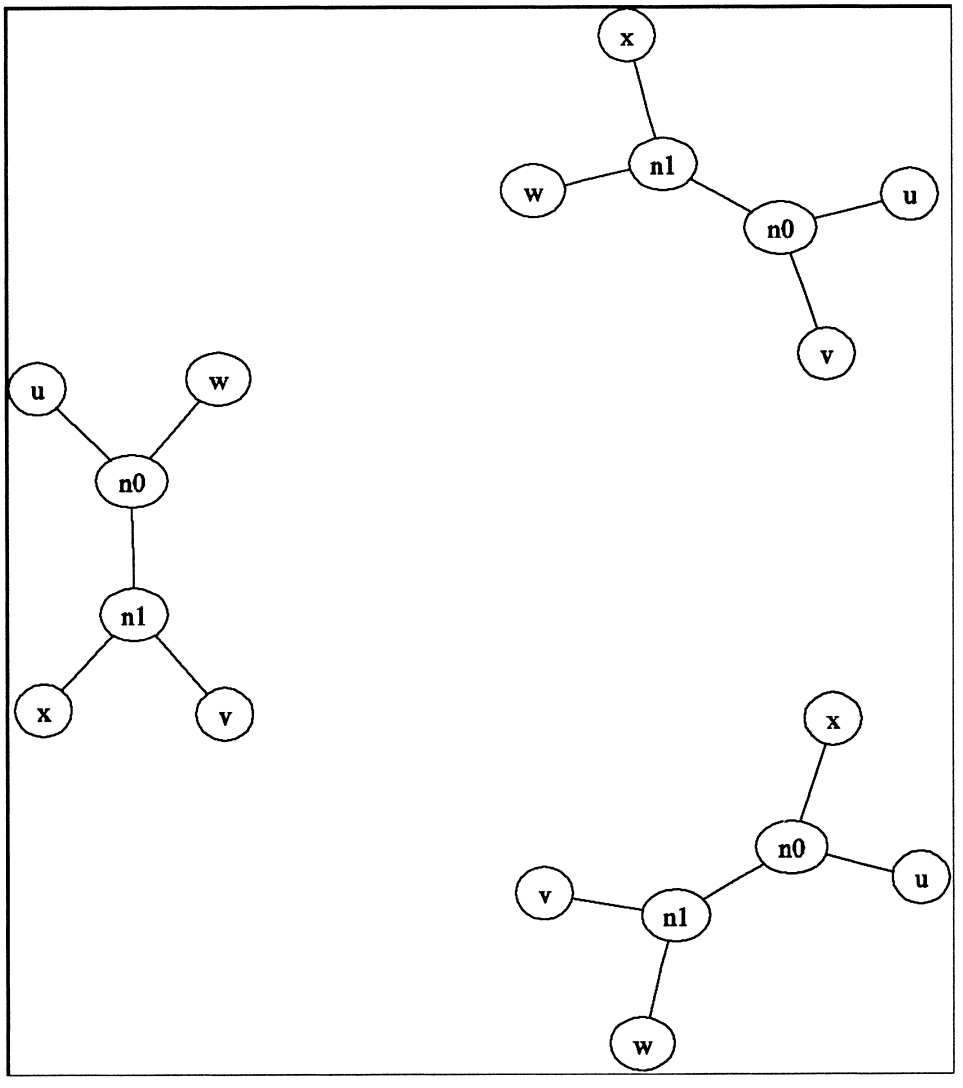
The *cost* of a quartet is defined as the sum of the distances between each pair of neighbors; that is, $C_{uv|wx} = d(u,v) + d(w,x)$. For any given tree T and any group of four leaf labels $\{u,v,w,x\}$, we say T is *consistent* with $uv|wx$ if and only if the path from u to v does not cross the path from w to x. Note that exactly one of the three possible quartets for any set of four labels must be consistent for any given tree. We may think of a large tree as having many smaller quartet trees embedded within its structure (see Figure 2). The total cost of a large tree is defined to be the sum of the costs of all consistent quartets. Below, we describe a heuristic method, developed first for the experiments reported here, that searches for a tree of minimal total cost.

Let us be more precise about the measure that our search optimizes. First, we generate a list of all possible quartets for all groups of labels under consideration. For each group of three possible quartets for a given set of four labels, we calculate a best (i.e., minimal) cost, and a worst (i.e., maximal) cost. Adding all the best quartets yields the best (i.e., minimal) cost, and adding all the worst quartets yields the worst (i.e., maximal) cost. The minimal and maximal values need not be attainable by actual trees, but the score of any tree will lie between these two values. 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 we call this the *normalized tree benefit score* $S(T)$. If $S(T) = 1$, then the tree T incorporates the best quartet for every four labels and hence represents the distance matrix without any distortion. (The representation is in a certain sense qualitative, because the tree edges do not represent true distances.) Roughly speaking, the closer $S(T)$ is to 1, the less distortion there is of the information in the distance matrix.

Experiments in this article as well as our article at arxiv.org/abs/cs.CV/0312044 show that it accords well with our intuitive judgment of the quality of a tree. The distance matrix resulting from n objects from a natural data set can be faithfully represented in n-dimensional Euclidean space. But when it is represented in lower-dimensional spaces, or in ternary trees, we incur unavoidable distortion, much like the projection of the spherical Earth's surface onto a flat map. As long as n is small, say $n \leq 15$, this distortion is low as is consistently evidenced by very high experimental $S(T)$ values. Also, confidence is high in that all randomized runs end up with the same tree. When n gets large, say $n \geq 35$, we noticed a decline in $S(T)$ value to below 0.9, whereas randomized runs end up with some varieties of trees.

The goal of the quartet method is to find a full tree with a maximum value of $S(T)$, which is to say, the lowest total cost. This optimization problem is known to be NP-hard (Jiang, Kearney,

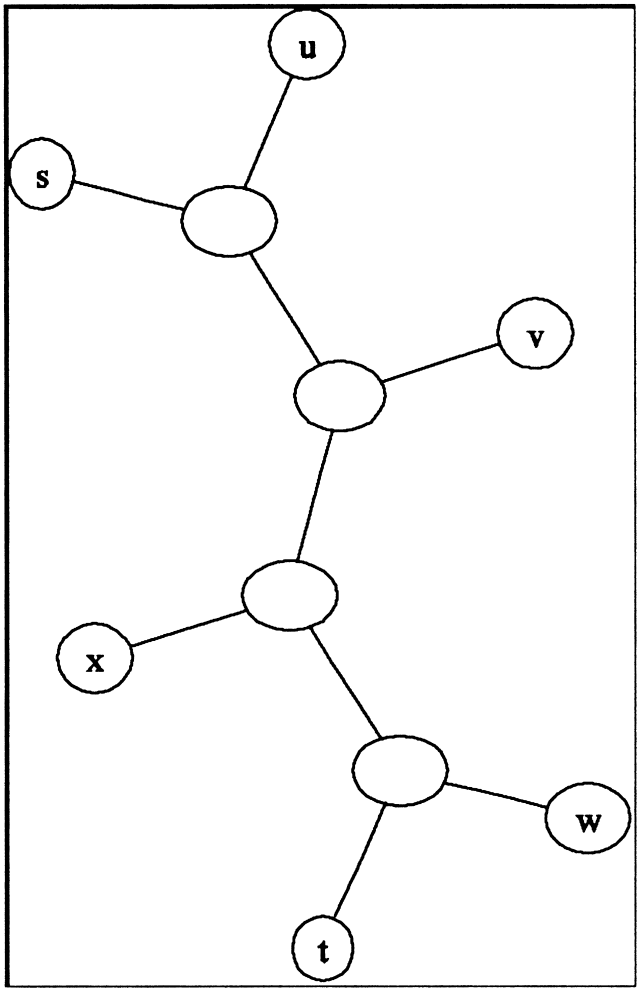
Figure 1. The three possible quartets for the set of leaf labels {u,v,w,x}.



and Li 2001), which means that it is infeasible in practice, but we can sometimes solve it and always approximate it. The current methods in Bryant et al. (2000) are far too computationally intensive; they run many months or years on moderate-sized problems of 30 objects.

We have designed a simple method based on randomization and hill-climbing. First, a random tree with $2n - 2$ nodes is created, consisting of n leaf nodes (with one 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 one-third 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

Figure 2. An example tree consistent with quartet uv|wx.



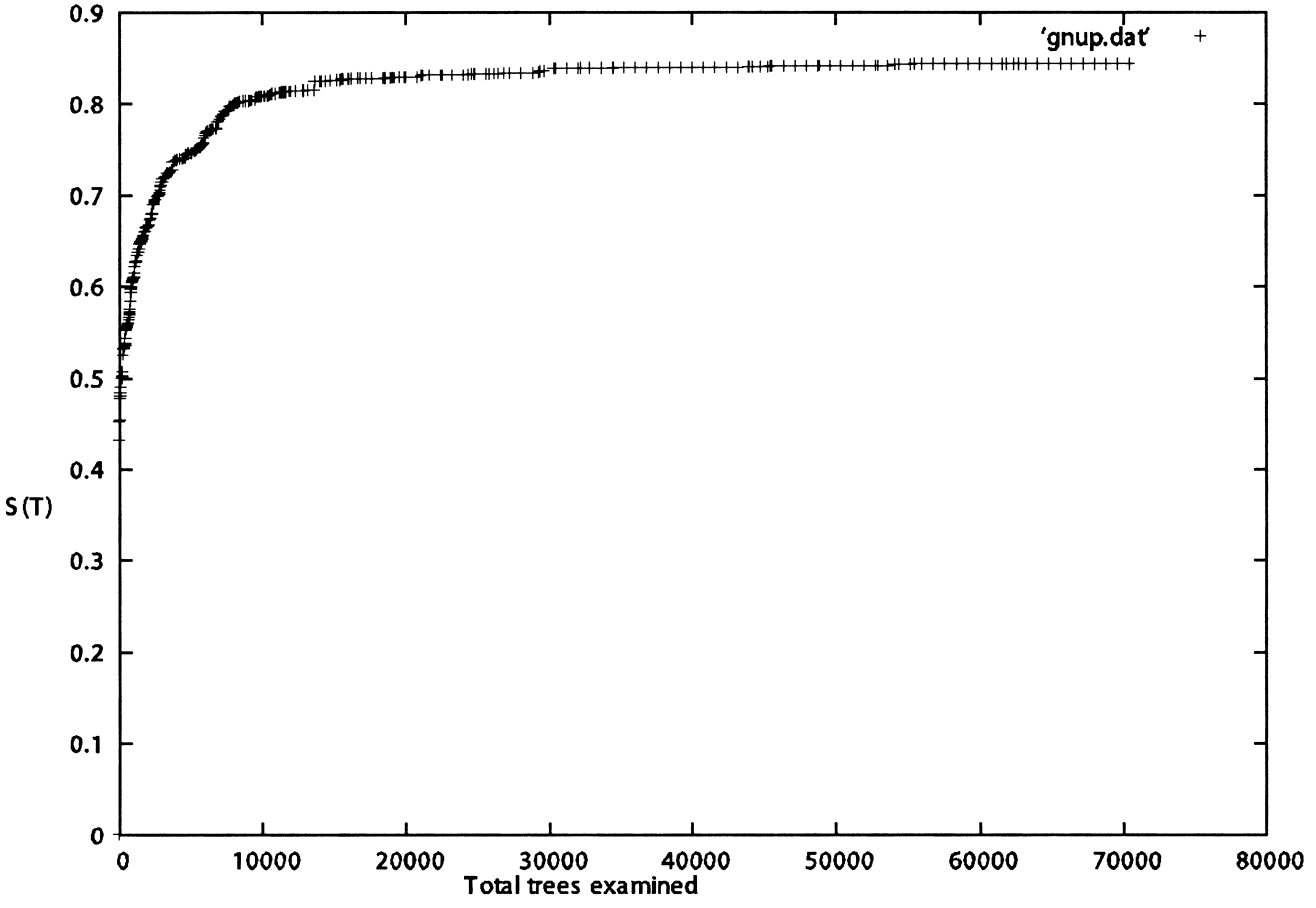
nodes; and (3) a *subtree transfer*, whereby a randomly chosen subtree (possibly a leaf) is detached and reattached in another place, maintaining similarity invariants.

Each of these simple mutations keeps invariant the number of leaf and internal nodes in the tree; only the structure and placements change. We 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 of mutations listed above with equal probability. Finally, for each of these simple mutations, we pick leaves or internal nodes as necessary. Notice that trees that are close to the original tree (in terms of the number of simple mutation steps in between) are examined often, and trees that are far away from the original tree will eventually be examined, but not very frequently. Thus, 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 we keep T' as the new best tree. Otherwise, we try a new different tree and repeat. If $S(T')$ ever reaches 1, then we halt and give the best tree as output. Otherwise, we 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. 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 35 objects, this tree-search gives a tree with $S(T) \geq 0.9$ within half an hour on a 1.5-GHz Pentium computer. 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. We can consider the graph of Figure 3, mapping the achieved $S(T)$ score as a function of the number of trees examined. Progress occurs typically in a sigmoidal fashion towards a maximal value $S(T) \leq 1$.

A problem exists, however. For natural data sets, we often see some leaf nodes (i.e., data items) placed near the center of the tree as singleton leaves attached to internal nodes without sibling leaf nodes. This results in a more linear, stretched-out, and less-balanced tree. Such trees, even if they represent the underlying distance matrix faithfully, are difficult to fully understand and may cause misunderstanding of represented relations and clusters.

Figure 3. Progress of the 60-piece experiment over time.



To counteract this effect, and to bring out the clusters of related items more visibly, we have added a penalty term of the following form: For each internal node with exactly one leaf node attached, the tree's score is reduced by 0.005. This induces a tendency in the algorithm to avoid producing degenerate mostly linear trees in the face of data that are somewhat inconsistent, and it creates balanced and more illuminating clusters. It should be noted that the penalty term causes the algorithm in some cases to settle for a slightly lower $S(T)$ score than it would have without penalty term. Also, the value of the penalty term is heuristically chosen. The largest experiment used 60 items, and we typically had only a couple of orphans, causing a penalty of only a few percent. This should be set off against the final $S(T)$ score of above 0.84. Another practicality concerns the stopping criterion, at which $S(T)$ value we stop. Essentially, we stopped when the $S(T)$ value did not change after examining a large number of mutated trees. An example is the progress shown in Figure 3.

Details of Our Implementation

The software that we developed for the experiments of this article and for later experiments reported online is freely available at complearn .sourceforge.net. For the experiments reported here, 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 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 only MIDI Note-On and Note-Off events. These events were then converted to a player-piano style representation, with time quantized in 0.05-sec 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 MIDI Note-On 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, producing a single signed 8-bit value as output 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 (BWT; Burrows and Wheeler 1994). 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 1.5-GHz Pentium computers (with 1 GB of RAM that we do not use).

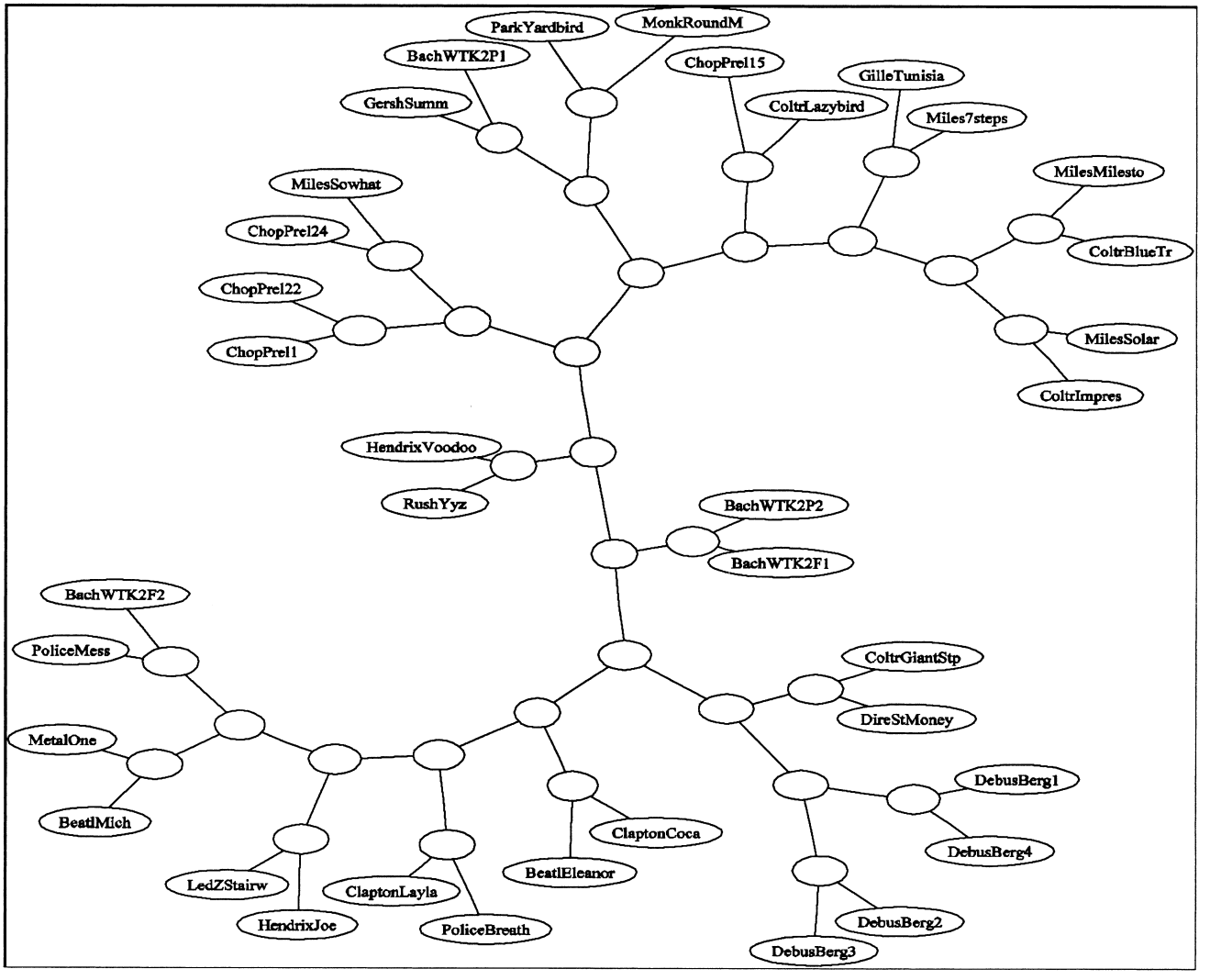
Results of Five Experiments

Genres: Rock vs. Jazz vs. Classical

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. All musical pieces we used are listed in the tables in the Appendix. For the genre-identification experiment, we used twelve classical pieces (the small set from Table 1, consisting of Bach, Chopin, and Debussy), twelve jazz pieces (Table 2), and twelve rock pieces (Table 3). The tree that our program produced is given in Figure 4. The $S(T)$ score is 0.858.

Viewed from our preconception about the genres, the discrimination between the three genres is good but not perfect. But already, the relatively low $S(T)$ value shows that the tree distorts the proximity relations represented by the distance matrix. Rerunning this experiment gave consistently about the same results, showing that the NCD distance matrix of this natural data set cannot be represented in ternary tree representation without significant distortion. The upper-right branch of the tree contains ten of the twelve jazz pieces, but also Chopin's Prélude No. 15 and a Bach Prelude. The

Figure 4. Output for the 36 pieces from three genres.



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 twelve rock pieces, nine are placed close together in the lower left branch, whereas Jimi Hendrix's ''Voodoo Chile,'' Rush's ''Yyz,'' and Dire Straits's ''Money for Nothing'' are further away. Most of the classical pieces are in the middle and lower-right part of the tree. The four Debussy movements are close together, as are three of the four Chopin Préludes.

Surprisingly, two of the four Bach pieces are rather misplaced. It is not clear why this happens and may be considered an error of our distance measure, because intuitively we perceive the four Bach pieces to be very close. On the other hand, the fact that they are closer to other pieces than to each other does reflect some (dis)similarity. In effect, our similarity engine aims at the ideal of a perfect data-mining process, discovering known as well as unknown features in which the data can be similar. Whether this specific placement of the Bach pieces reveals anything about musical similar-

ity remains to be seen; it could also be some similarity that is an artifact of our model. Indeed, ideally some of the more surprising similarities claimed by our method could point the way for further musicological investigation.

Classical Piano Music (Small Set)

For the next three experiments, we restrict attention to clustering classical piano pieces. In Table 1, we list all 60 pieces used, together with their abbreviations. Some of these are complete compositions; others are individual movements from larger compositions. Before running our program on the whole set of 60 piano pieces, we first tried it on two smaller sets: a small twelve-piece set, indicated by "(s)" in Table 1, and a medium-size 32-piece set indicated by "(s)" or "(m)." The small set encompasses the four movements from Debussy's *Suite Bergamasque*, four movements of Book 2 of Bach's *Das Wohltemperierte Klavier*, and four Préludes from Chopin's Opus 28. As one can see in Figure 5, our program does a fairly good job at clustering these pieces. The $S(T)$ score of 0.958 is also high. Because the number of objects is small, the tree can represent the NCD matrix of this natural data set without much distortion.

The four Debussy movements form one cluster, as do the four 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 three Chopin pieces. This Prélude No. 15 ("Raindrop"), in fact, consistently forms an odd one out in our other experiments as well. There may be some musical truth to this, as No. 15 is probably the most exceptional among the 24 Préludes of Chopin's Opus 28. For example, it is by far the longest, it exhibits the most variety in dynamics, and it includes an unusually high incidence of repetition of a single pitch.

Classical Piano Music (Medium Set)

The medium set adds 20 pieces to the small set: six additional Bach pieces, six additional Chopin pieces, one more Debussy piece, and seven pieces by Haydn. The experimental results are given in Figure 6. The $S(T)$ score of 0.895 is slightly lower than in the small-set experiment. Again, there is a lot of structure and expected clustering. Most of the Bach pieces are together, as are the four Debussy pieces from the *Suite Bergamasque*. The fifth Debussy item is somewhat apart, which is not too surprising as it comes from another piece. Both the Haydn and the Chopin pieces are clustered in little sub-clusters of two to four pieces, but those subclusters are scattered throughout the tree instead of being close together in a larger cluster.

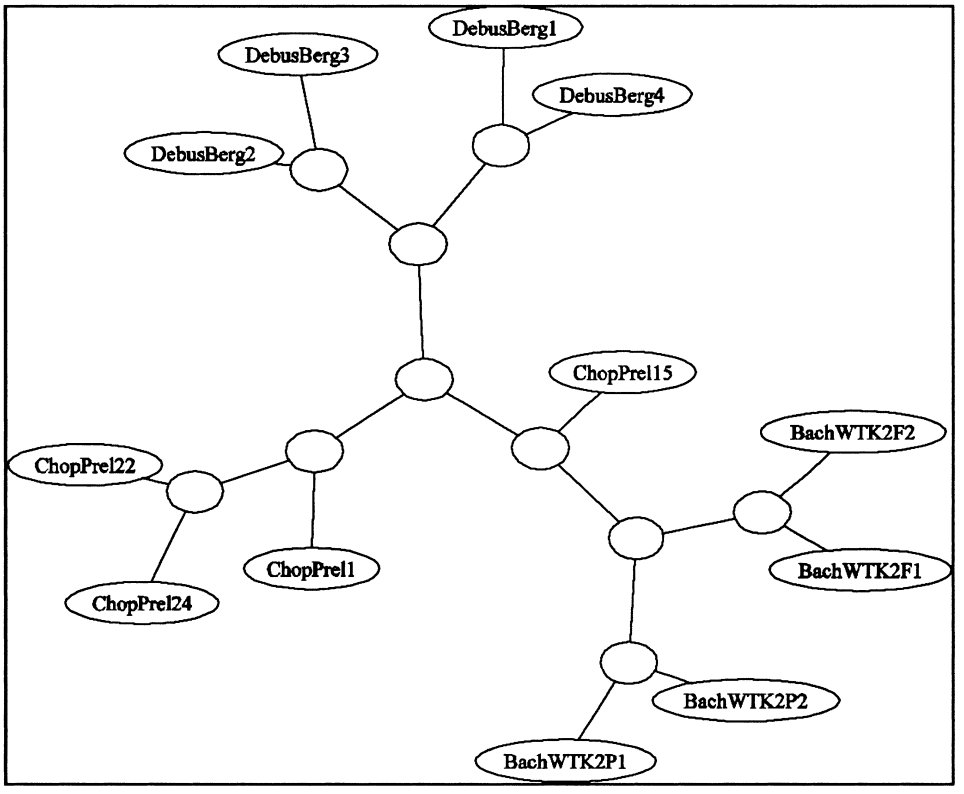
Classical Piano Music (Large Set)

Figure 7 gives the output of a run of our program on the full set of 60 pieces. This experiment adds ten pieces by Beethoven, eight by Buxtehude, and ten by Mozart to the medium set. The experimental results are given in Figure 7. The results are still far from random but leave more to be desired than the smaller-scale experiments. Indeed, the $S(T)$ score has dropped further from that of the medium-sized set to 0.844, implying increasing distortion of the proximity relations in the NCD matrix. Bach and Debussy are still reasonably well clustered, but other pieces, notably the Beethoven and Chopin, are scattered throughout the tree. The placement of the pieces is closer to intuition on a small level (for example, most pairing of siblings corresponds to musical similarity in the sense of the same composer) than on the larger level. This is similar to the phenomenon of little sub-clusters of Haydn or Chopin pieces that we saw in the medium-set experiment.

Clustering Symphonies

Finally, we tested whether the method worked for more complicated music, namely 34 symphonic pieces. We took two Haydn symphonies (No. 95 in one file, and the four movements of No. 104), three Mozart symphonies (Nos. 39, 40, and 41), three

Figure 5. Output for the 12-piece set.



Beethoven symphonies (Nos. 3, 4, and 5), Schubert's *Unfinished Symphony*, and three movements of Saint-Säens's Symphony No. 3.

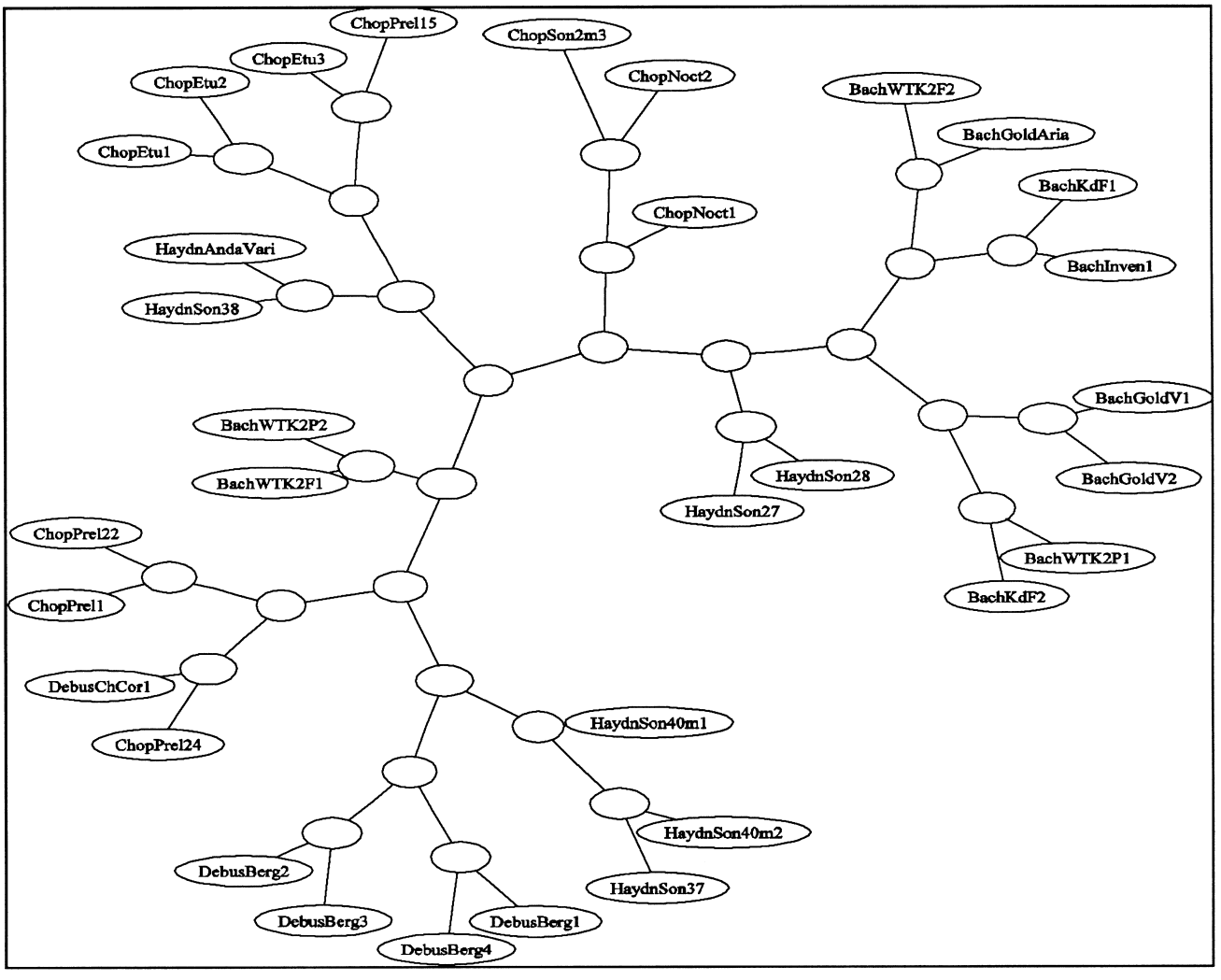
The results are given in Figure 8, with a passable $S(T)$ score of 0.860. Broadly speaking, the Mozart movements are in the lower part of the tree and the Beethoven movements are in the upper and upper-left part. Three of the four movements from Haydn's Symphony No. 104 are in the upper-right part, two of the three Saint-Säens movements are together, whereas the two Schubert movements look more or less randomly placed. It is interesting to note that our method is more likely to cluster together similarly structured movements from different symphonies than different movements from the same symphony. This suggests that, for example, two allegro movements from different Mozart symphonies have more in common than the allegro and adagio movements from the same Mozart symphony.

Conclusions and Future Work

In this article, 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 suboptimal because the incomputable Kolmogorov complexity must be replaced by some computable approximation—in this case, a real-world compressor (we used bzip2 here).

We described various experiments where we first computed the matrix of pairwise distances between the various MIDI files involved, and then we used a

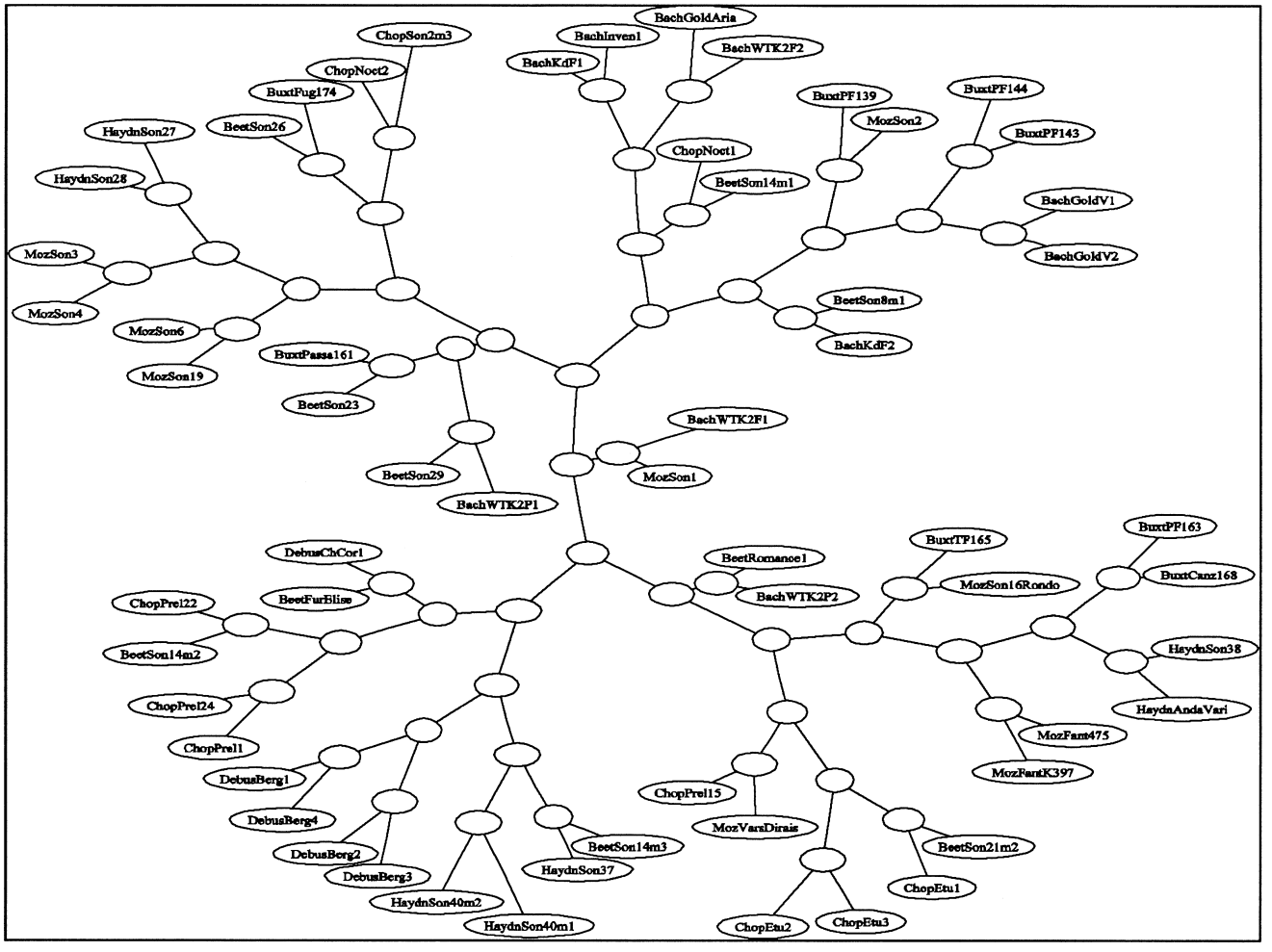
Figure 6. Output for the 32-piece set.



new heuristic tree construction algorithm to assemble 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. We view this as a strength of the method, though one may also view it as a limitation, because it makes it more difficult to bring useful musical background knowledge into play to improve the results. The versatility and general-purpose nature of our method is also exemplified by the range of later experiments reported in our online article (arxiv.org/abs/cs.CV/0312044), and the robustness is evidenced by the consistency of results under variations of compressors used.

Our research raises many questions worth examining further. For one, it is difficult to judge the results of our experiments in a more rigorous way than just to see whether the tree looks acceptable intuitively (i.e., is congruent with our intuitive expectations about musical similarity). Our $S(T)$ score quantitatively measures how well a tree conforms to a matrix of distances, but not really how well the tree conforms to our expectations. This also

Figure 7. Output for the 60-piece set.



makes it difficult to compare our results with other methods.

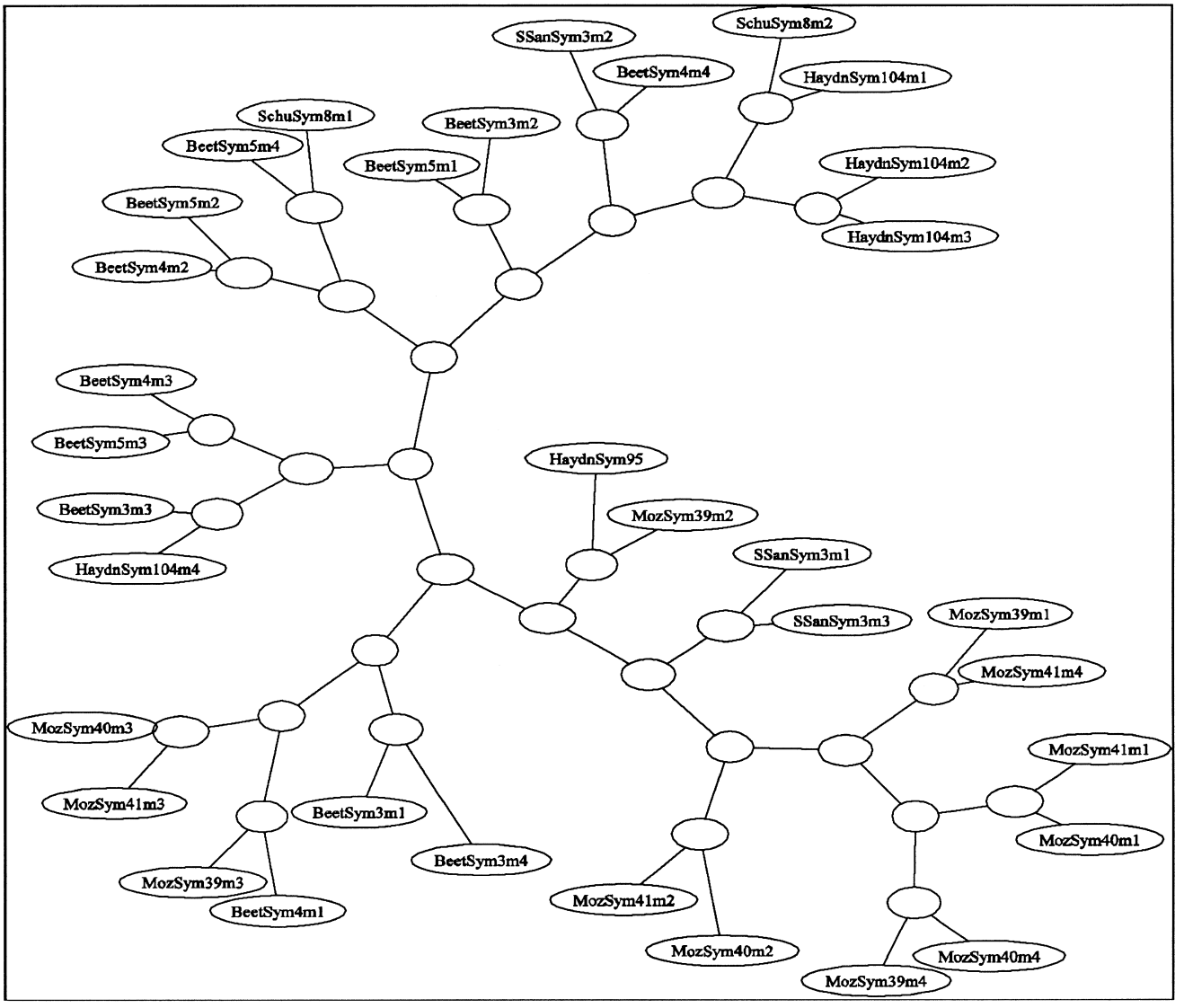
Second, the program can be used as a data-mining machine to discover hitherto unknown similarities between music pieces of different composers or different genres. In this manner, we can discover plagiarism or indeed honest influences between music pieces and composers. It is possible that we can use the method to discover seminality of composers, or separate music eras and fads.

A very interesting and third application of our program would be to select a plausible composer for a newly discovered piece of music of which the composer is not known. In addition to such a piece, this experiment would require a number of pieces from known composers that are plausible candidates. We would run our program on the set of all those pieces and see where the new piece is placed. If it lies squarely within a cluster of pieces by a particular composer, then that would be a plausible candidate composer for the new piece.

A fourth area to examine lies in the quartet method. Each run of our program is different—even on the same set of data—owing to our use of randomness for choosing mutations in the quartet method. Experiments have shown that the higher the $S(T)$ score, the more stable the outcomes are over different such runs.

Fifth, at various points in our program, somewhat arbitrary choices were made. Examples are

Figure 8. Output for the set of 34 movements of symphonies.



the compression algorithms used (all practical compression algorithms will fall short of Kolmogorov complexity, but some compress better than others); the way we transform the MIDI files (choice of time-interval length, choice of note representation); and the cost function in the quartet method. Other choices are possible and may or may not lead to better clustering. For example, we compared the quartet-based approach to the tree reconstruction with alternatives. One such alternative that we tried is to compute the Minimum Spanning Tree (MST) from the matrix of distances. MST has the advantage of being very efficiently computable, but it resulted in trees that were much worse than the quartet method. It appears that the quartet method is extremely sensitive in extracting information even from small differences in the entries of the distance matrix, where other methods would be led to error. Ideally, one would like to have well-founded theoretical reasons to decide such choices

in an optimal way. Lacking those, trial-and-error seems the only way to deal with them.

Finally, the experimental results got decidedly worse when the number n of pieces grew, apparently owing to the inherently greater distortion of representing n-dimensional metric distances in a ternary tree for larger n.

Acknowledgments

We thank John Tromp and Mike Paunovich for useful discussions, and the editors of *Computer Music Journal* and anonymous referees for pointers to the literature and for many comments that improved the presentation of this article.

This work was supported in part by NWO, the EU project RESQ, IST-2001-37559, the NoE QUIPROCONE +IST-1999-29064, the ESF QiT Programme, and the EU Fourth Framework BRA NeuroCOLT II Working Group EP 27150.

References

Ball, P. 2002. "Algorithm Makes Tongue Tree." Available at www.nature.com/nsu/020121/020121-2.html.

Benedetto, D., E. Caglioti, and V. Loreto 2002. "Language Trees and Zipping." *Physical Review Letters* 88(4):048702–1-048702-4.

Bennett, C., et al. 1998. "Information Distance." *IEEE Transactions on Information Theory 44*(4):1407–1423.

Bryant, D., et al. 2000. "A Practical Algorithm for Recovering the Best Supported Edges of an Evolutionary Tree." *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*. New York: Association for Computing Machinery, pp. 287–296.

Burrows, M., and D. J. Wheeler. 1994. "A Block-Sorting Lossless Data Compression Algorithm." Technical Report 124. Palo Alto, California: Digital Equipment Corporation.

Chai, W., and B. Vercoe. 2001. "Folk Music Classification Using Hidden Markov Models." Paper presented at the International Conference on Artificial Intelligence, 4–10 August, Seattle, Washington.

Cilibrasi, R. 2003. "The CompLearn Toolkit." Available online at complearn.sourceforge.net.

Cormode, G., et al. 2000. "Communication Complexity of Document Exchange." *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*. New York: Association for Computing Machinery, pp. 197–206.

Dannenberg, R., B. Thom, and D. Watson. 1997. "A Machine-Learning Approach to Musical Style Recognition." *Proceedings of the 1997 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 344–347.

Dubnov, S., et al. 2003. "Using Machine-Learning Methods for Musical Style Modeling." *Computer* 36(10):73–80.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification*, 2nd ed. Hoboken, New Jersey: Wiley.

Ghias, A., et al. 1995. "Query by Humming: Musical Information Retrieval in an Audio Database." *Proceedings of the 1995 ACM Multimedia Conference*. New York: Association for Computing Machinery, pp. 231–236.

Grimaldi, M., A. Kokaram, and P. Cunningham. 2002. "Classifying Music by Genre Using the Wavelet Packet Transform and a Round-Robin Ensemble." Technical Report TCD-CS-2002-64. Dublin, Ireland: Trinity College.

Jiang, T., P. Kearney, and M. Li. 2001. "A Polynomial Time Approximation Scheme for Inferring Evolutionary Trees from Quartet Topologies and Its Application." *SIAM Journal of Computing* 30(6):1942–1961.

Li, M., et al. 2001. "An Information-Based Sequence Distance and Its Application to Whole Mitochondrial Genome Phylogeny." *Bioinformatics* 17(2):149–154.

Li, M., et al. 2003. "The Similarity Metric." *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*. New York: Association for Computing Machinery, pp. 863–872.

Li, M., and P. Vitányi. 1997. *An Introduction to Kolmogorov Complexity and Its* Applications, 2nd ed. New York: Springer-Verlag.

Li, M., and P. Vitányi. 2002. "Algorithmic Complexity." In N. J. Smelser and P. B. Baltes, eds. *International Encyclopedia of the Social and Behavioral Sciences*. Oxford: Pergamon, pp. 376–382.

Londei, A., V. Loreto, and M. O. Belardinelli. 2003. "Musical Style and Authorship Categorization by Informative Compressors." *Proceedings of the 5th Triennial ESCOM Conference*. Liege, Belgium: European Society for the Cognitive Sciences of Music, pp. 200–203.

Orpen, K., and D. Huron. 1992. "Measurement of Similarity in Music: A Quantitative Approach for Non-Parametric Representations." *Computers in Music Research* 4:1–44.

Tzanetakis, G., and P. Cook. 2002. "Music Genre Classification of Audio Signals." *IEEE Transactions on Speech and Audio Processing* 10(5):293–302.

Appendix: The Music Used

Tables 1–3 list all pieces of music used in our study. Table 1 gives the 60 classical pieces used, Table 2 gives the twelve jazz pieces, and Table 3 gives the twelve rock pieces.

Table 1. The 60 Classical Pieces Used

Composer	*Piece*	*Acronym*
J. S. Bach (10)	*Das Wohltemperierte Klavier II*: Preludes and Fugues 1, 2	BachWTK2{F,P}{1,2} (s)
	Goldberg Variations: Aria, Variations 1 and 2	BachGold{Aria,V1,V2} (m)
	Die Kunst der Fuge: Variations 1 and 2	BachKdF{1,2} (m)
	Invention 1	BachInven1 (m)
Beethoven (10)	Sonata No. 8 (''Pathetique''), first movement	BeetSon8m1
	Sonata No. 14 (''Mondschein''), three movements	BeetSon14m{1,2,3}
	Sonata No. 21 (''Waldstein''), second movement	BeetSon21m2
	Sonata No. 23 (''Appassionata'')	BeetSon23
	Sonata No. 26 (''Les Adieux'')	BeetSon26
	Sonata No. 29 (''Hammerklavier'')	BeetSon29
	Romance No. 1	BeetRomance1
	Für Elise	BeetFurElise
Buxtehude (8)	Preludes and Fugues, BuxWV 139, 143, 144, and 163	BuxtPF{139,143,144,163}
	Toccata and Fugue, BuxWV 165	BuxtTF165
	Fugue, BuxWV 174	BuxtFug174
	Passacaglia, BuxWV 161	BuxtPassa161
	Canzonetta, BuxWV 168	BuxtCanz168
Chopin (10)	Préludes Op. 28: 1, 15, 22, and 24	ChopPrel{1,15,22,24} (s)
	Études Op. 10, Nos. 1, 2, and 3	ChopEtu{1,2,3} (m)
	Nocturnes Nos. 1 and 2	ChopNoct{1,2} (m)
	Sonata No. 2, third movement	ChopSon2m3 (m)
Debussy (5)	*Suite Bergamasque*, four movements	DebusBerg{1,2,3,4} (s)
	Children's Corner Suite (''Gradus ad Parnassum'')	DebusChCor1 (m)
Haydn (7)	Sonatas No. 27, 28, 37, and 38	HaydnSon{27,28,37,38} (m)
	Sonata No. 40, movements 1 and 2	HaydnSon40m{1,2} (m)
	Andante and Variations	HaydnAndaVari (m)
Mozart (10)	Sonatas No. 1, 2, 3, 4, 6, and 19	MozSon{1,2,3,4,6,19}
	Rondo from Sonata No. 16	MozSon16Rondo
	Fantasias KV397, KV475	MozFantK{397,475}
	Variations on ''Ah, vous dirais-je madam''	MozVarsDirais

(m) = presence in the medium set; (s) = presence in the small and medium sets.

Table 2. The Twelve Jazz Pieces Used

John Coltrane	''Blue Trane''
	''Giant Steps''
	''Lazy Bird''
	''Impressions''
Miles Davis	''Milestones''
	''Seven Steps to Heaven''
	''Solar''
	''So What''
George Gershwin	''Summertime''
Dizzy Gillespie	''Night in Tunisia''
Thelonious Monk	''Round Midnight''
Charlie Parker	''Yardbird Suite''

Table 3. The Twelve Rock Pieces Used

The Beatles	''Eleanor Rigby''
	''Michelle''
Eric Clapton	''Cocaine''
	''Layla''
Dire Straits	''Money for Nothing''
Led Zeppelin	''Stairway to Heaven''
Metallica	''One''
Jimi Hendrix	''Hey Joe''
	''Voodoo Chile''
The Police	''Every Breath You Take''
	''Message in a Bottle
Rush	''Yyz''