Estimating Resemblance of MIDI Documents

Michael Mitzenmacher[⋆] and Sean Owen[⋆⋆]

Computer Science Department, Harvard University.
{michaelm, srowen}@eecs.harvard.edu.

Abstract. Search engines often employ techniques for determining syntactic similarity of Web pages. Such a tool allows them to avoid returning multiple copies of essentially the same page when a user makes a query. Here we describe our experience extending these techniques to MIDI music files. The music domain requires modification to cope with problems introduced in the musical setting, such as polyphony. Our experience suggests that when used properly these techniques prove useful for determining duplicates and clustering databases in the musical setting as well.

1 Introduction

The extension of digital libraries into new domains such as music requires rethinking techniques designed for text to determine if they can be appropriately extended. As an example of recent work in the area, Francu and Nevill-Manning describe designing an inverted index structure for indexing and performing queries on MIDI music files [8]. Their results suggest that by using additional techniques such as pitch tracking, quantization, and well-designed musical distance functions, organizing and querying a large music database can be made efficient.

In this paper, we describe our experience extending hashing-based techniques designed for finding near-duplicate HTML Web documents to the problem of finding near-duplicate MIDI music files. These techniques are currently used by the AltaVista search engine to, for example, determine if an HTML page to be indexed is nearly identical to another page already in the index [6]; they have also been used to cluster Web documents according to similarity [7]. Our goal is to determine if these techniques can be effective in the setting of musical databases. This follows a trend in other recent work in the area of musical databases in trying to extend text-based techniques to musical settings [4,8].

As an example of related work, the Humdrum Toolkit [1] provides a suite of tools that may be used to estimate musical resemblance and manipulate musical structures in other ways. Indeed, the techniques we describe could be implemented in Humdrum, and we believe they may prove a useful addition to the

[⋆] Supported in part by an Alfred P. Sloan Research Fellowship, NSF CAREER Grant CCR-9983832, and an equipment grant from Compaq Computer Corporation.

[⋆⋆] Supported in part by a grant from the Harvard Committee for Faculty Research Support.

toolkit. We note that Humdrum uses its own proprietary format (although translations to and from MIDI are possible). Themefinder [2] implements a musical database search tool using the same format as Humdrum and allows searches on a melody line. We instead allow comparison between entire pieces of music.

Our results suggest that finding near-duplicates or similar MIDI pieces can be done efficiently using these hashing techniques, after introducing modifications to cope with the additional challenges imposed by the musical setting. In particular we address the problem of polyphony, whereas other work on musical similarity has largely restricted itself to single, monophonic melody lines. Applications for these techniques are similar to those for Web pages, as described in [7]. We focus on the most natural applications of clustering and on-the-fly resemblance computation.

To begin, we review important aspects of the MIDI file format and recall the basic framework of these hashing techniques in the context of text documents. We then discuss the difficulties in transferring this approach to the musical domain and suggest approaches for mitigating these difficulties. We conclude with results from our current implementation.

1.1 Review: MIDI

The MIDI (Musical Instrument Digital Interface) 1.0 specification [3] defines a protocol for describing music as a series of discrete note-on and note-off events. Other information, such as the force with which a note is released, volume changes, and vibrato can also be described by the protocol. A MIDI file provides a sequence of events, where each event is preceded by timing information, describing when the event occurs *relative to the previous event*. MIDI files therefore describe music at the level that sheet music does. Because of its simplicity and extensibility, MIDI has become a popular standard for communication of musical data, especially classical music.

1.2 Review: Hashing Techniques for Text Similarity

We review previously known techniques based on hashing for determining when text documents are syntactically similar. Here we follow the work of Broder [6], although there are other similar treatments [9,10].

Each document may be considered as a sequence of words in a canonical form (stripped of formatting, HTML, capitalization, punctuation, etc.). A contiguous subsequence of words is called a *shingle*, and specifically a contiguous subsequence of w words is a *w-shingle*. For a document A and a fixed w there is a set $S(A, w)$ corresponding to all w-shingles in the document. For example, the set of 4-shingles $S(A, 4)$ of the phrase "one two three one two three one two three" is:

{(one, two, three, one), (two, three, one, two), (three, one two, three)}

(We could include multiplicity, but here we just view the shingles as a set.)

One measure of the resemblance of two text files A and B is the resemblance of their corresponding sets of shingles. We therefore define the resemblance $r(A, B)$ as:

$$r(A, B) = \frac{|S(A, w) \cap S(B, w)|}{|S(A, w) \cup S(B, w)|}$$

The resemblance $r(A, B)$ is implicitly dependent on w, which is generally a prechosen fixed parameter. The resemblance is a number between 0 and 1, with a value of 1 meaning that the two documents have the same set of w-shingles. Small changes in a large document can only affect the resemblance slightly, since each word change can affect at most w distinct shingles. Similarly, resemblance is resilient to changes such as swapping the order of paragraphs.

An advantage of this definition of resemblance is that it is easily approximated via hashing (or fingerprinting) techniques. We may hash each w-shingle into a number with a fixed number of bits, using for example Rabin's fingerprinting function [5]. From now on, when using the term shingle, we refer to the hashed value derived from the underlying words. For each document, we store only shingles that are 0 modulo p for some suitable prime p. Let $L(A)$ be the set of shingles that are 0 modulo p for the document A. Then the estimated value of the resemblance r_e is given by:

$$r_e(A, B) = \frac{|L(A) \cap L(B)|}{|L(A) \cup L(B)|}$$

This is an unbiased estimator for the actual resemblance $r(A, B)$. By choosing p appropriately, we can reduce the amount of storage for $L(A)$, at the expense of obtaining possibly less accurate estimates of the resemblance. As $L(A)$ is a smaller set of shingles derived from the original set, we call it a *sketch* of the document A.

Similarly, we may define the containment of A by B, or $c(A, B)$, by:

$$c(A, B) = \frac{|S(A, w) \cap S(B, w)|}{|S(A, w)|}$$

Again, containment is a value between 0 and 1, with value near 1 meaning that most of the shingles of A are also shingles of B. In the text setting, a containment score near 1 suggests that the text of A is somewhere contained in the text of B. We may estimate containment by:

$$c_e(A, B) = \frac{|L(A) \cap L(B)|}{|L(A)|}$$

Again this is an unbiased estimator.

2 Challenges in Adapting Hashing Techniques to Music

2.1 Polyphony

The greatest challenge in adapting hashing techniques similar to those for text described above is that while text is naturally represented as a sequence of bytes, musical notes occur logically in parallel, not serially. Thus musical data formats like MIDI must arbitrarily flatten this data into a serial stream of note events. This creates several potential problems related to *polyphony*, or the playing of several notes concurrently, so that directly converting the MIDI representation into a text file and applying the above techniques is not sufficient. Much of the work in musical resemblance has obviated this problem by considering only monophonic sequences of notes like a simple melody. We attempt to deal with music in its entirety.

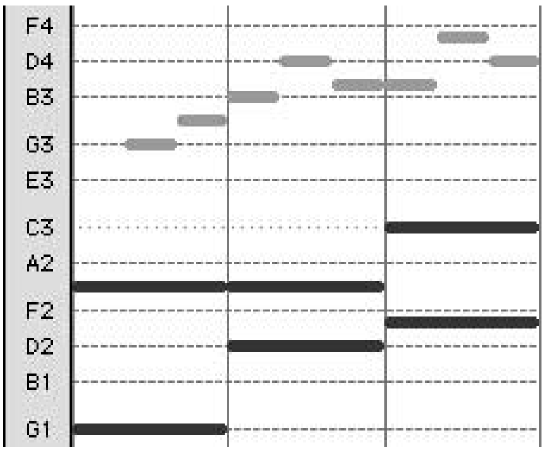

Fig. 1. Excerpt from J.S. Bach's "Jesu, Joy of Man's Desiring"

For example, Figure 1 shows three measures of J.S. Bach's "Jesu, Joy of Man's Desiring" in roll-bar notation. A sequence of note events representing this excerpt might begin as follows:

G1 On, **G2 On**, G3 On, G3 Off, A3 On, **G1 Off**, **G2 Off**, A3 Off, **D2 On**, **G2 On**...

The events representing the bass notes are boldfaced; they are interspersed among the melody note events, although intuitively the two are logically separate sequences. As this example demonstrates, a logical group of notes such as the melody usually *is not* represented contiguously in a MIDI file, because note events occurring in other parts come in between.

This situation confounds attempts to derive meaningful resemblance estimates from direct application of text resemblance techniques. For example, if one used the natural text representation of a MIDI file to compute the resemblance or containment of some MIDI file and another MIDI file containing only the melody of the same piece, one could not expect to correctly identify the similarity between the files. Adjacent events in a MIDI file may not have any logical relationship, so grouping them into shingles is no longer meaningful in the context of a MIDI file.

2.2 Timing and Other Information

Since MIDI records event timing with fine resolution, the timing of note events in two MIDI representations of the same music could differ, but to an imperceptible degree. Such insignificant variations can potentially have a significant impact on a resemblance estimate. MIDI files may also record a variety of musical meta-information and machine-specific controls not directly related to the music.

3 Solutions

3.1 Pitch by Pitch

Our basic strategy for adapting to musical data is to separate out notes from each pitch, and fingerprint the sequences of start times for notes of each pitch independently. That is, a separate sketch is produced for each of the 128 possible note pitches in a MIDI file. Contiguous subsequences of note start times in each pitch are grouped into shingles, and so forth as in the text resemblance computation described above. So, the resemblance of C3 note events in document A with C3 note events in document B is computed by direct application of the text resemblance computation, and likewise for all 128 possible pitches. A weighted average of these 128 resemblance computations then gives the resemblance between A and B.

This helps mitigate the problems of polyphony in both resemblance and containment computations. The notes in one pitch are more likely to belong to *one* logical group of notes, therefore it is reasonable to group notes of the same pitch together. At the level of a single pitch, the text resemblance computation is again meaningful and we can take advantage of this established technique. We may eliminate the problem of arbitrary ordering of simultaneous events; when considering one pitch, simultaneous events are exact duplicates, and one can be ignored.

One may ask why we do not group notes from two or three adjacent pitches together, or group notes across all 128 pitches together into a single sequence. Our initial experimental results indicate that this hurts performance, and indeed grouping all pitches together yields extremely poor performance. While this approach may be worth further study, we believe the advantages of the pitch by pitch approach in the face of polyphony are quite strong.

3.2 Extension to Transpositions and Inversions

The pitch by pitch approach also allows us to consider musical transpositions. Consider a musical piece in the key of C in MIDI file A, and the same piece in the key of C# in file B. As given, this resemblance computation would return a very low resemblance between A and B, though musically they are all but identical. C3 notes in A are compared to C3 notes in B, but this group of notes really corresponds to C#3 notes in B.

If we were to account for this by comparing C#3 notes in B to C3 notes in A, and so forth, we would correctly get a resemblance of 1.0. This is trivial using pitch by pitch sketches; we may try all possible transpositions and take the maximum resemblance as the resemblance between two MIDI files. While this certainly solves the problem, it slows the computation substantially. (There are up to 128 transpositions to try.) In practice we may try the few most likely transpositions, using information such as the number of shingles per pitch to determine the most likely transpositions.

Grouping by pitch into 128 sequences is not the only possibility. For example, another possibility is to *ignore the octave of each pitch entirely*; all C pitches would be considered identical and all C note events would be grouped, altogether producing 12 sequences. Doing so potentially creates the same interference effects between parts that we try to avoid, however.

On the positive side, ignoring the octave helps cope with *harmonic inversions*. For example, the pitches C, E, and G played together are called a C major chord, regardless of their relative order. That is, C3-E3-G3 is a C major chord, as is E3-G3-C4, though the latter is said to be “in inversion.” All such inversions have the same subjective harmonic quality as the original (“root position”) chord.

If one made some or all the C3 notes in a piece into C4 notes (moving them all up by one octave), the resulting piece would be subjectively similar, yet these two pieces would be deemed different by the computation described above. Ignoring octaves clearly resolves this problem.

These examples demonstrate the difficulty involved in both adequately defining and calculating musical similarity at a syntactic level. In our experimental setup, we choose not to account for transpositions or harmonic inversions. Our domain of consideration is classical MIDI files on the web, and such variations are generally uncommon; that is, one does not find renditions of Beethoven’s “Moonlight Sonata” transposed to any key but its original C# minor. However one could imagine other domains where such considerations would be important.

3.3 Timing

In text, the word is the natural base for computation. In music, the natural base is the note. In particular, we have chosen the relative timings of the notes as the natural structure, corresponding to the order of words for text documents.

We have found that using only the start time, instead of start time and duration, to represent a note is the most effective. The duration, or length of time a note lingers, can vary somewhat according to the style of the musician, as well as other factors; hence it is less valuable information and we ignore it. Even focusing only on start times, various preprocessing steps can make the computation more effective.

MIDI files represent time in “ticks,” a quantity whose duration is typically 1/120th of a quarter-note, but may be redefined by MIDI events. We scale all time values in a MIDI file as a preprocessing step so that one tick is 1/120th of a quarter-note. Given this and the fact that we ignore MIDI Tempo events, the same music with a different tempo should appear similar using our metric.

All times are quantized to the nearest multiple of 60 ticks, the duration of an eighth-note (typically between 150 and 400 milliseconds) in order to filter out small, unimportant variations in timing.

Recall that note start times are given relative to previous notes. Musically speaking, a time difference corresponding to four measures is in a sense the same as forty measures, in that they are both a long period of time, probably separating what would be considered two distinct blocks of notes ("phrases"). We therefore cap time differences at 1,920 ticks, which corresponds to 4 measures of 4/4 time (about 8 seconds), and do not record any shingle containing a time difference larger than this. Another option is to cap start times at 1,920 ticks but not discard any shingles.

3.4 Extraneous Information

MIDI files may contain a great deal of information besides the note-on and note-off events, *all of which we consider irrelevant to the resemblance computation.* This is analogous to ignoring capitalization and punctuation in the text domain; note that we could include this additional information, but we choose to ignore it for overall performance.

For example, we ignore information about the instruments used and the author and title of the piece, and base the computation solely on the music. In practice such information could be used for indexing or classifying music, see e.g. [4]. We note here that one could certainly use this information in conjunction with our techniques, although it implies that one trusts the agents generating this information.

Similarly we ignore track and channel numbers. MIDI files may be subdivided into multiple tracks, each of which typically contains the events describing one musical part (one instrument, possibly) in a complex musical piece. Track divisions are ignored because they have no intrinsic musical meaning and are not employed consistently. Notes from all tracks in a multi-track MIDI file are merged into a single track as a preprocessing step. Also, MIDI events are associated with one of 16 channels to allow for directed communication on a network; this channel number also has no musical meaning and is ignored.

Musically, we ignore differences in tempo, volume, and note velocity, focusing on the timing aspects as described above.

3.5 The Algorithm

Fingerprinting. For each pitch z, the note events in pitch z in a MIDI file are extracted and viewed as a sequence of numbers (start times). Times are normalized, and recall that start times are recorded as the time since the last event. Each subsequence of four start times is viewed as a shingle; any shingle containing a start time larger than 1,920 ticks is discarded. Each shingle is hashed using Rabin fingerprints [5] into a 16-bit number; those shingles whose hash is 0 modulo 19 are kept in the sketch of pitch z for the file.

Note that increasing the number of start times per shingle decreases the likelihood of false matches but magnifies the effect of small variations on the resemblance score. We have found experimentally that four start times per shingle is a reasonable compromise, although the choice is somewhat arbitrary. Three start times per shingle gives poorer results, but five and even six start times per shingle yield performance quite similar to that of four.

The modulus 19 may be varied to taste; smaller values can increase accuracy of resemblance estimates at the cost of additional storage, and vice versa.

A 16-bit hash value economizes storage yet yields a somewhat small hash space. Since an average MIDI file produces only about 1,200 shingles to be hashed, the probability of collision is small enough to justify the storage savings. Note that using 20-bit or 24-bit hash values introduces substantial programming complications that will slow computation; therefore 16 bits appears the best practical choice, because using a full 32-bit hash value would substantially increase memory requirements.

Resemblance and Containment. Let A_z be the sketch (set of shingles) for pitch z in a file A. Given sketches for two files A and B, for each z we compute the resemblance

$$r_z = \frac{|A_z \cap B_z|}{|A_z \cup B_z|},$$

just as in the text resemblance computation. A weighted average of these 128 values gives the resemblance between A and B, where r_z's weight is $(|A_z|+|B_z|)$. Other scaling factors are possible; this scaling factor is intuitively appealing in that it weights the per pitch resemblance the total number of shingles. In particular, this approach gives more weight to pitches with many matching notes, which is useful in cases where one file may only contain certain pitches (such as only the melody). We have found this performs well in practice, and in particular it performs better than using an unweighted average.

Containment is defined analogously; the containment score for pitch z is instead weighted by $|A_z|$.

4 Experimental Results

4.1 Test Data

To our knowledge there is no large, publicly available standard corpus of MIDI files. Instead we developed our own corpus using five sites offering MIDI files on the Internet. We downloaded 14,137 MIDI files (11,198 of which were unique) for testing purposes. Many sites limit downloads to 100 per user per day, so custom robot programs were developed to obtain the MIDI files while respecting usage restrictions.

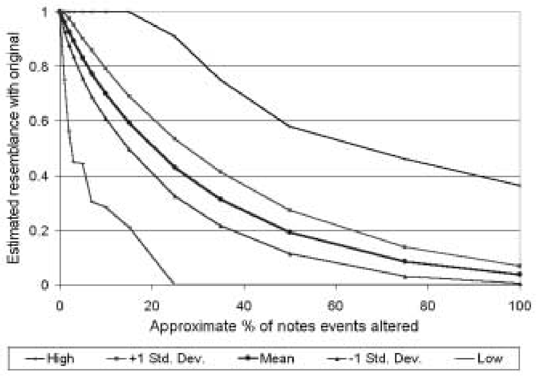

Fig. 2. Percent of note events altered vs. estimated resemblance

4.2 Behavior of the Resemblance Score

A musical resemblance computation is only useful insofar as it corresponds well to some intuitive notion of resemblance. That is, an estimated resemblance should allow reliable conclusions about the relationship between two MIDI files. Our proposed computation corresponds to the notion that two files resemble each other when they have a high proportion of note events in common. Other notions of resemblance are possible, but this is probably the most natural and is also the notion captured by the text resemblance computation.

To test the accuracy of this computation, we took 25 MIDI files from the web and generated 100 altered MIDI files from each of them. Each note event was changed with probability $q/100$, so roughly a percentage q of the note events were slightly altered at random. If a note was changed, then:

– with probability 1/4 its pitch was changed, either up or down by one pitch, at random.
– with probability 1/4 it was deleted.
– with probability 1/2 it was given a new start time, chosen uniformly at random between the original start time and the time of either the last event or an implicit event 120 ticks before, whichever was more recent.

Although these changes do not correspond to a specific model of how differences in MIDI files might arise, we feel it provides a reasonable test of the system's performance.

The resemblance of each of the 2,500 instances was computed with the original. The distribution of the resulting resemblance scores versus q is shown in Figure 2. There is a reasonable and fairly reliable relationship between q and the resulting resemblance score.

To view the results in a different way, we also consider a small test set consisting of five familiar pieces. The pieces are Beethoven's "Moonlight Sonata," First Movement; Saint-Saëns's "Aquarium" from "Carnival of the Animals;"

Rimsky-Korsakov's "Flight of the Bumblebee;" Wagner's "Bridal Chorus from Lohengrin;" and Schumann's "Traumerei." Variations on each of these five files were constructed, with 3%, 6%, and 9% of all notes altered. Also, files consisting of only the treble parts of the songs were created. As Table 1 shows, there can be significant variance in the results depending on the piece.

Table 1. Sample resemblance computations

	Moonlight	Aquarium	Flight	Lohengrin	Traumerei
Treble	0.9375	0.4968	0.7739	0.8000	0.6667
3%	0.9179	0.8850	0.8657	1.0000	1.0000
6%	0.7050	0.7529	0.7087	0.8793	0.8571
9%	0.6463	0.7412	0.5474	0.7829	0.4583

4.3 Simple Clustering

This resemblance computation's most compelling application may be determining when two MIDI files represent the same piece of music. We clustered our corpus of documents in the following way: any pair of files A and B for which $r(A, B)$ exceeded some threshold t were put into the same cluster.

We find that high thresholds ($t > 0.45$) all but eliminate false matches; the contents of nearly every cluster correspond to the same piece of music, though *all* renditions of one piece may fail to cluster together. It is very interesting that variations of the same piece of music can have a fairly low resemblance score. (See the discussion for Table 2.) This appears to be a major difference between the musical setting and the text setting; there appears to be a wide variation in what constitutes the same musical piece, while for text the syntactic differences among what is considered similar is much less. It may also reflect a partial problem with the pitch by pitch approach: a delay in one note can affect the relative timing of multiple pitches, so changes can have an effect on a larger number of shingles.

At low thresholds ($t < 0.15$), with high accuracy all renditions of a piece of music cluster together, but different musical pieces often cluster together also. For instance, many (distinct) Bach fugues tend to end up the same cluster because of their strong structural and harmonic similarities. For such values of t we find that a few undesirably large clusters of several hundred files form; many small clusters aggregate because of a few fluke resemblances above the threshold, and snowball into a meaningless crowd of files.

To gain further insight into the importance of the threshold, we did pairwise resemblance comparisons for all the files in our corpus. In Figure 3 we show the number of pairs with a given resemblance; this is a histogram with the resemblance scores rounded to the nearest thousandth. (Notice the y-axis is on a logarithmic scale.) The graph naturally shows that most documents will have low resemblance; even moderate resemblance scores may be significant.

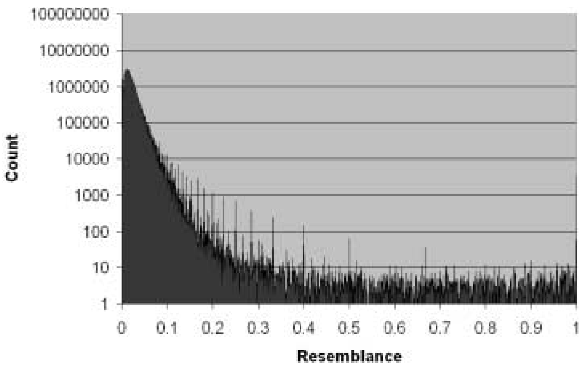

Fig. 3. Relative frequency of resemblance scores from our corpus.

We chose a compromise value of $t = 0.35$, for which we find that clusters tend to correspond well to distinct pieces of music with very few false matches; we present some of the results of a clustering with this threshold. Table 2 shows the site's descriptive information for the contents of some representative clusters. Identical renditions (by the same sequencer), as well as renditions from different sequencers cluster together consistently.

Such a clustering might help the Classical MIDI Connection (CMC) learn that their "Prelude No. IV in C# min" is from *The Well-Tempered Clavier*, or point out that the attribution of "Schubert-Liszt Serenade" to Liszt is possibly incorrect. This might also help a performer or MIDI site determine who else has posted (perhaps illegitimately) their own MIDI files on the web. As an example of the importance of the choice of threshold, consider the cluster corresponding to the variations of "The Four Seasons." The pairwise resemblances of its members are shown in Table 3. The two variations by the sequencer Dikmen appear identical, so we consider the first four from the same source but from different sequencers. As can be seen, the same piece of music can yield very different shingles from different sequencers.

Naturally, more sophisticated clustering techniques could improve performance. For example, we have also tried incrementally building clusters, putting clusters for A and B together only when the average resemblance of A with all members of B's cluster, and vice versa, exceeds some threshold. We find that this eliminates the problematic large clusters described above, but otherwise yields nearly identical clusters, and that a lower threshold of $t = 0.2$ performs well.

4.4 Performance

Source code, written in C, was compiled and run on a Compaq AlphaServer DS20 with a 500 MHz CPU, four gigabytes of RAM, a 64 KB on chip cache and a 4 MB on board cache.

Our implementation can fingerprint MIDI data at a rate of approximately 7.8 megabytes per second (of user time). If a typical MIDI file is around 45 kilobytes in size, then this amounts to producing sketches for about 174 MIDI files per second. A typical MIDI file's sketch requires about 128 bytes of storage, not counting bookkeeping information such as the file's URL. Approximately 3,096 resemblances can be computed per second; this may be sped up by faster I/O, by sampling fewer shingles (that is, increasing p), or by searching in parallel. We expect that performance could also be improved by rewriting our prototype code.

Table 2. Contents of some representative clusters

File Description	**Sequencer**	**Source**
Prelude No. 4 from *The Well-Tempered Clavier*, Book I (Bach)	(unknown)	CMC
Prelude No. IV in C# min (Bach)	(unknown)	CMC
Prelude No. 4 in C# min from *The Well-Tempered Clavier*, Book I (Bach)	M. Reyto	prs.net
Variations on a Theme by Haydn (Brahms)	J. Kaufman	CMC
Variations for Orchestra on a Theme from Haydn's St. Anthony's Chorale (Brahms)	J. Kaufman	prs.net
The Four Seasons, No. 2 - 'Summer' in G-, Allegro non molto (Vivaldi)	M. Dikmen	prs.net
The Four Seasons, No. 2 - L'Estate (Summer) in G-, 1. Allegro non molto (Vivaldi)	M. Reyto	prs.net
The Four Seasons, No. 2 - 'Summer' in G-, 2. Estate (Vivaldi)	A. Zammarrelli	prs.net
The Four Seasons, No. 2 - 'Summer' in G-, Allegro non molto (Vivaldi)	N. Sheldon Sr.	prs.net
Summer from the Four Seasons	M. Dikmen	Classical MIDI
Symphony No. 3 in D (Op. 29), 2nd Mov't (Tchaikovsky)	S. Zurflieh	CMC
Symphony No. 3, 2nd Mov't (Tchaikovsky)	S. Zurflieh	prs.net
Symphony No. 3 Op. 29, Movt. 2 (Tchaikovsky)	S. Zurflieh	sciortino.net
Schwanengesang, 4. Serenade (Schubert)	F. Raborn	prs.net
Schubert-Liszt Serenade (Liszt)	(unknown)	Classical MIDI
Serenade (Schubert)	F. Raborn	Classical MIDI
Symphony No. 94 in G 'Surprise,' 4. Allegro molto (Haydn)	J. Urban	prs.net
Symphony No. 94 in G 'Surprise,' 4. Finale: Allegro molto (Haydn)	L. Jones	prs.net

Source	**URL**
CMC	`http://www.midiworld.com/cmc/`
prs.net	`http://www.prs.net/midi.html`
Classical MIDI	`http://www.classical.btinternet.co.uk/page7.htm`
sciortino.net	`http://www.sciortino.net/music/`

Table 3. Similarity scores from a cluster.

	Dikmen	**Reyto**	**Zammarrelli**	**Sheldon**
Dikmen		0.4636	0.3233	0.3680
Reyto	0.4636		0.3612	0.5901
Zammarrelli	0.3233	0.3612		0.2932
Sheldon	0.3680	0.5901	0.2932	

5 Conclusions

We believe that this musical resemblance computation represents an effective adaptation of established text resemblance techniques to the domain of MIDI files. The pitch by pitch fingerprinting approach provides a useful and sound framework for this adaptation, and can be easily extended to tackle more complex musical issues like transpositions and inversions. Our experiments suggest that computation can be used to discover near-duplicate MIDI files with a high degree of accuracy. Further engineering and tuning work would be useful to optimize this approach.

We believe this approach may be useful in conjunction with other techniques for organizing and searching musical databases. An important open question is how these techniques can be applied to other musical formats, such as MP3.

References

1. The Humdrum Toolkit. http://www.lib.virginia.edu/dmmc/Music/Humdrum/.
2. Themefinder. Available at http://www.themefinder.org.
3. MIDI Manufacturers Association. The complete detailed MIDI 1.0 specification, 1996.
4. D. Bainbridge, C. G. Nevill-Manning, I. H. Witten, L. A. Smith, and R. J. McNab. Towards a digital library of popular music. In *Proceedings od Digital Libraries '99*, pages 161-169, 1999.
5. A. Z. Broder. Some applications of Rabin's fingerprinting method. In R. Capocelli, A. De Santis, and U. Vaccaro, editors, *Sequences II: Methods in Communications, Security, and Computer Science*, pages 143–152. Springer-Verlag, 1993.
6. A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES '97)*, pages 21–29. IEEE Computer Society, 1998.
7. A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proceedings of the Sixth International World Wide Web Conference*, pages 391–404, 1997.
8. C. Francu and C. G. Nevill-Manning. Distance metrics and indexing strategies for a digital library of popular music. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, 2000.
9. U. Manber. Finding similar files in a large file system. In *Proceeding of the Usenix 1994 Winter Technical Conference*, pages 1–10, January 1994.
10. N. Shivakumar and H. Garcia-Molina. SCAM: A copy detection mechanism for digital documents. In *Proceeding of the 2nd International Conference in the Theory and Practice of Digital Libraries (DL '95)*, 1995.