The MUSART Testbed for Query-by-Humming Evaluation

Roger B. Dannenberg,* William P. Birmingham,† George Tzanetakis,†† Colin Meek,§ Ning Hu,* and Bryan Pardo¶
*School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 USA
{roger.dannenberg, ning.hu}@cs.cmu.edu
†Math and Computer Science Department
Grove City College
Grove City, Pennsylvania 16127 USA
wpbirmingham@gcc.edu
††Computer Science Department
University of Victoria
Victoria, British Columbia V8W3P6 Canada
gtzan@cs.uvic.ca
§SQL Server
Microsoft Corporation
Redmond, Washington 98052 USA
cmeek@microsoft.com
¶Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109-2110 USA
bryanp@umich.edu

Music Information Retrieval has become an active area of research motivated by the increasing importance of Internet-based music distribution. In December 2003, Apple Computer announced it was selling almost 1.5 million music downloads per week (www.apple.com/pr/library/2003/dec/15itunes.html), and some analysts predict that downloads will account for 33 percent of the music industry's sales by 2008 (Zeidler 2003). Online catalogs are already approaching one million songs, so it is important to study new techniques for searching these vast stores of audio.

One approach to finding music that has received much attention is Query-by-Humming (QBH). This approach enables users to retrieve songs and information about them by singing, humming, or whistling a melodic fragment. In QBH systems, the query is a digital audio recording of the user, and the ultimate target is a complete digital audio recording. The audio waveforms of the query will have little or no direct similarity to those of the target audio recording, so QBH systems always search using some other representation. Most commonly, this representation is a sequence of notes described by pitch and duration. It is possible to transcribe monophonic queries into note sequences (although accurate transcription of the monophonic voice is still an active research area). Polyphonic target music, however, cannot be automatically transcribed into melodies. Therefore, most QBH systems assume that a MIDI or symbolic representation is available from which a note sequence can be derived.

Our system uses a database consisting of standard MIDI files, so one can state the QBH problem as follows: ''Given a user's audio query, find matching melodies in a database of standard MIDI files.'' Once the melody is identified, the QBH system might offer links to audio files, ring tones, album titles, card catalog information, sheet music, or other useful data.

While many researchers have investigated related problems and have built prototype systems, there have been few systematic investigations of different solutions. It is difficult to compare different studies, because there are no standard databases adopted by the community of researchers, and in-

Computer Music Journal, 28:2, pp. 34–48, Summer 2004
© 2004 Massachusetts Institute of Technology.

tellectual property issues inhibit the sharing of music to form a standard database. Our work aims to evaluate different techniques for QBH using a common framework, the MUSART testbed, so that we can obtain quantitative information about the relative performance of different search techniques.

We hope to work with other researchers to compare other systems in the future. One conclusion of our work is that merely reporting data such as ''90 percent of the time, the correct song is included in the 10 closest matches'' is not very meaningful. Performance is highly sensitive to the quality of the queries and the material in the database. Therefore, it is important to compare search algorithms using a common set of queries and data. We have compared three classes of algorithms in this manner.

Another result of our work is a method for studying how retrieval precision is affected by the size of the database. Because it is difficult to construct a large music database for research, we would like to have some idea of how performance will scale as databases grow large. We present some evidence that retrieval rates fall very slowly as the database size increases. This trend is simple to compute and could be useful to predict performance for any retrieval system.

In the next section, we describe our project, the motivation, and the origins of this research in more detail, followed by the software architecture of our testbed. Then, we describe three different search systems that we have studied. Next, we describe the performance of these search systems in our testbed, followed by the general sources of error we observed that limit QBH performance and a discussion of searching performance as databases grow. Finally, we present a general discussion and conclusions.

The MUSART Project

The MUSART project is a collaboration between the University of Michigan and Carnegie Mellon University. Together, we have been exploring the design of QBH systems (Birmingham et al. 2001; Hu and Dannenberg 2002; Meek and Birmingham 2002a; Pardo and Birmingham 2002; Shifrin et al. 2002). We have developed a variety of algorithms based on hidden Markov models and contour matching. In addition, we have implemented several versions of note-sequence-matching algorithms using dynamic programming.

As our research progressed, it became expedient for project members to adopt their own data and methods. As we developed and implemented search algorithms, we also created new signal-analysis software, collected new queries, added files to our databases, and improved our theme-extraction software. With so many variables, it was simplest to hold constant a collection of data and programs in order to focus on one or two experimental variables.

After following these procedures for a year or two, we found it increasingly difficult to compare systems. They had simply become incompatible. We feel that this state of affairs in our microcosm mirrors the state of the field in general (Downie 2002; Futrelle and Downie 2002). Many results have been published (Ghias et al. 1995; McNab et al. 1996; Pauws 2002), but evaluation is difficult, and results are not comparable.

To remedy this situation, at least in our own research project, we created a general testbed that is capable of hosting all our work on content-based retrieval. The testbed includes collections of queries, target data, analysis software, and search algorithms. Another testbed system is described by Bainbridge, Dewsnip, and Witten (2002), and, like this study, their work evaluates different algorithms within a consistent framework. We have integrated several of our research systems into our testbed and are able to compare the systems objectively. Some of our data can be shared, and we can also evaluate algorithms for other researchers using our testbed.

The Testbed Architecture

The MUSART testbed is hosted on a Linux server and relies on scripts written in Python to conduct experiments. The use of Python makes it easily portable to other operating systems. Our goal is for complete tests to run from start to finish without manual intervention. A typical test begins with a collection of audio queries, a database of target MIDI files, and a variety of programs to process audio, process MIDI, and search the database. The output of a test includes statistical information about the search results in text and graphical plot formats. All input and output data can be viewed using a Web browser so that researchers (in Pennsylvania, Michigan, and Washington) can have convenient access to all results from all tests.

To support different systems, including various preprocessing stages, we adopted the model shown in Figure 1. In this model, the input to the system consists of queries (generally an audio recording of someone singing a melody) and targets (generally MIDI files to be searched). We have a number of collections of queries and targets, which we store in a hierarchical directory structure. For any given test run, we describe the queries and targets of interest as lists of filenames. This allows us to reproduce our results even if new files are added to the database.

In addition to queries and targets, we have intermediate representations. Most search systems convert queries to transcriptions stored as MIDI files or to pitch contours stored as data in text files. We usually process standard MIDI file targets with the ThemeExtractor program (Meek and Birmingham 2001) to obtain short theme files to be searched. Our scripts will automatically generate these intermediate representations if possible. It is also possible to import intermediate representations as files when their construction is not fully automated.

Our system needs the correct target(s) for each query to evaluate search performance. Because there may be several versions of a song in the database, we keep a file for each query that lists all the correct targets. When reporting rank order, we report the lowest-ranking correct target.

Tests of search systems are saved in a "results" directory. Search programs take one query and a list of targets, and generate match scores indicating how well the query matches the target. The test script then collects the results and sorts them to calculate the rank order of the correct target. The script produces an easy-to-parse text output summary for further analysis. The output includes six items:

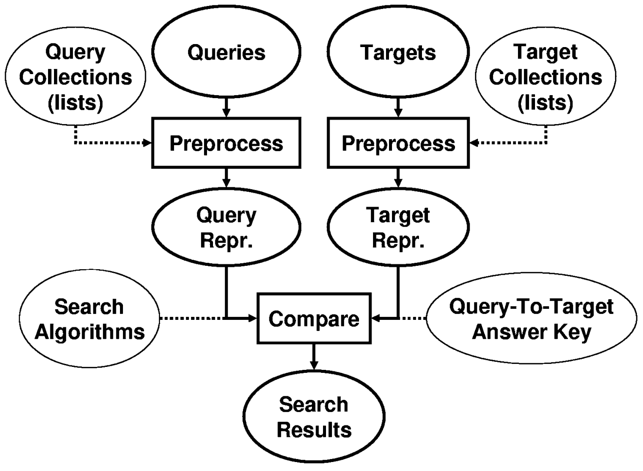


Figure 1. Architecture of the MUSART testbed.

- The name of the queries collection file
- The name of the targets collection file
- The search algorithm and any command-line options used
- The query preprocessor and any command-line options used
- The match score for each target and the rank order of the correct target for each query
- Statistical information (the mean rank, average deviation, standard deviation, and a histogram of ranks)

The output format assumes full searches in which the query is compared to every target in the database, but it would be relatively simple to change this assumption and return less information.

Description of Search Systems

The primary goal of our testbed is to enable objective comparisons between different search methods. We have focused on our three best-performing algorithms. The first applies dynamic programming string-matching algorithms to match sequences of pitch intervals and inter-onset interval (IOI) ratios. The second applies dynamic time-warping algorithms to compare melodic contours. The third uses a hidden Markov model to account for differences between queries and targets. We report here the results from the best configurations of our algorithms. With two-query transcription systems, two theme finders, and many variations in the search algorithms, the space of possibilities is quite large.

Query Transcription

Before describing the search algorithms, we will describe briefly our query-transcription and theme-extraction processes. Because queries are monophonic, our transcriber relies heavily on fundamental-frequency estimation. A common method for pitch estimation is autocorrelation (Roads 1996), which works on the principle that a quasi-periodic signal is highly correlated with itself when shifted by multiples of the fundamental period. Therefore, the autocorrelation typically shows a strong peak at the fundamental period. However, the autocorrelation often has false peaks when there are strong harmonics. We use the enhanced autocorrelation technique (Tolonen and Karjalainen 2000) in which a copy of the autocorrelation is time-stretched by a factor of two and subtracted from the original. This tends to suppress the strongest effects of harmonics and thereby reduces the false peaks, leading to more reliable fundamental-frequency estimates. Queries are processed using a window size of about 50 msec and a step size of 10 msec. For each 10-msec frame, we record a fundamental frequency estimate, or zero if the frame has either a low-amplitude or a low-autocorrelation peak.

One of our search algorithms, the Melodic-Contour search (described below), uses these fundamental frequency estimates fairly directly. The other search algorithms require that we derive a sequence of notes from the input query. Notes are recorded for any consecutive run of five or more non-zero pitch estimates (50 msec) that do not change by more than one semitone. To avoid any absolute pitch reference and to allow somewhat for pitch drift, the intervals between these notes are quantized to semitone intervals. Figure 2 shows a six-note sung sequence transcribed and segmented by the transcriber. Gray dots indicate instantaneous pitches generated by the transcriber; black bars indicate pitch-quantized and segmented notes.

Theme Extraction

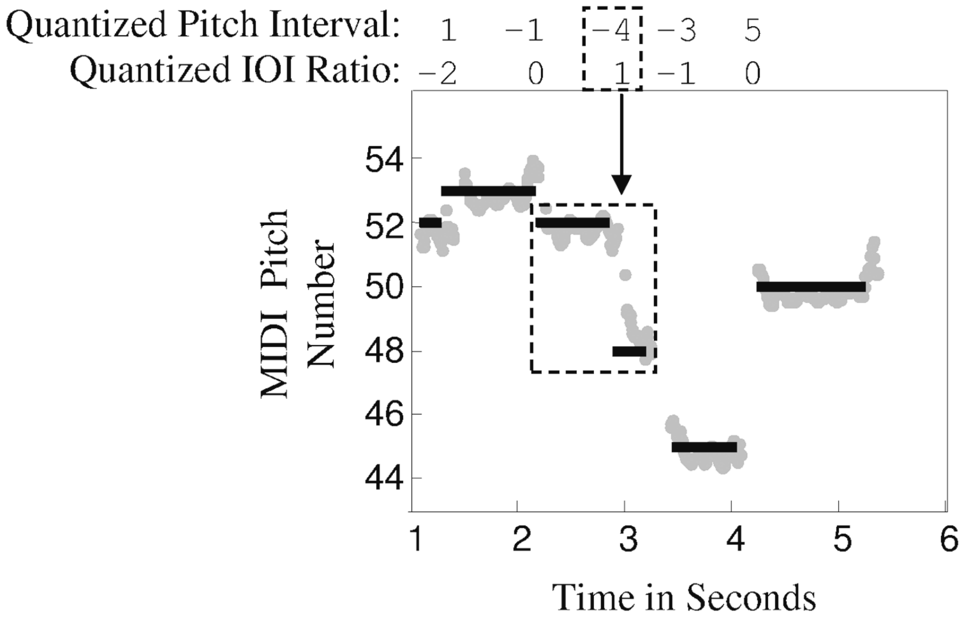
To optimize searching, the database should consist of melodies that users are likely to remember. Harmony lines, percussion lines, and other material are not as useful. Our database consists of MIDI files that we found on the World Wide Web, and most files include rhythm, harmony, and parts for at least several instruments. It would be very time-consuming to extract melodies from all these files by hand, so we use a program, ThemeExtractor, to locate themes automatically (Meek and Birmingham 2001). This system begins by identifying melodic patterns in a piece of music, and it then characterizes those patterns according to several features, including register, rhythmic consistency, and several ad hoc measures of the degree to which the pattern might be considered ''interesting.'' These features are then normalized and weighted to provide a score for each pattern. Using these scores, we return what are deemed to be the most significant phrases in the piece. By searching themes rather than full MIDI files, we hope to avoid spurious matches to ''musical filler'' and at the same time speed up our search process by condensing the size of the data to be searched.

The next three sections describe the three QBH search algorithms we evaluated in the MUSART testbed.

Note-Interval/Dynamic Programming Search

The first search algorithm is the Note-Interval or Dynamic Programming search. This system relies on segmentation by our query transcriber to estimate note onset times and pitches in audio queries. Both targets and queries are transformed into sequences of note intervals, each of which consists of a pitch interval and a rhythmic interval. Pitch intervals are quantized to the nearest half step and range from –12 to +12 half steps. Rhythmic intervals are represented as one of five logarithmically spaced inter-onset interval ratio (IOIr) values from –2 to +2 (Pardo and Birmingham 2002). The IOIr represents the ratio between the duration of the previous note and the current note.

Figure 2. Query transcription, pitch quantization, and note segmentation. Intervals are labeled above the graph with pitch interval (in semitones) and IOI ratios (on a quantized logarithmic scale).



Encoding a melody as note intervals is both tempo-invariant and transposition-invariant, letting us match queries at a variety of tempi and transpositions without time-stretching or transposing the targets. The values at the top of Figure 2 show the final encoding of the note sequence as duples of pitch interval and IOIr. The values (–4, 1) in the box represent the interval between the notes indicated by the other box in the figure.

Similarity between a target and the query is given by the minimum cost of transforming the target into the query using three editing operations: insert a note interval, delete a note interval, and substitute a note interval in the query for a corresponding one in the target (Pardo and Birmingham 2002; Pardo, Birmingham, and Shifrin 2004). Both insertion and deletion are fixed-cost operations. The reward (or cost) of substituting a query note-interval for a target note-interval is based on the similarity of the note intervals. Reward for substitution decreases monotonically with distance in either IOIr or pitch-interval (Pardo, Birmingham, and Shifrin 2004).

The highest-reward alignment is calculated by filling an alignment-matrix, a portion of which is shown in Figure 3. Notice that because each cell of the matrix depends only upon cells to the left and above, the entire matrix can be calculated in a single pass. This step takes time proportional to *mn*, the product of the lengths of the query and target interval sequences.

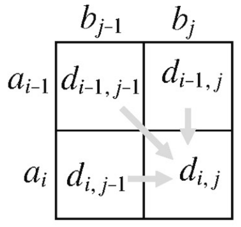
Because it is likely that there will be some significant overlap between the query and theme, but that each will have a beginning or end portion not covered by the other, we use the local string-matching algorithm described in Durbin et al. (1998), which allows a restart whenever the reward for an alignment drops below zero. The equation for the local alignment score matrix is given in Figure 3.

The match score between a target and query is the highest value in the alignment matrix for the pair. The match score is taken to be a direct measure of the similarity between target and query. Targets are ranked by match score, and the one with the highest score is deemed the best match to the query.

Melodic-Contour/Dynamic Time-Warping Search

The Melodic-Contour matcher is based on the idea that whereas pitch estimation is not too difficult,

Figure 3. Calculation of the best local alignment score for Note-Interval search.



$$d_{i,j} = \mathbf{max} \left\{ \begin{array}{c} 0 \\ d_{i-1,j-1} + substitutionReward(a_i, b_j) \\ d_{i-1,j} - skipCost_a \\ d_{i,j-1} - skipCost_b \end{array} \right\} \quad (1 \le i \le m,\ 1 \le j \le n)$$

segmentation into notes is very difficult and error-prone. A segmentation error corresponds to a note insertion or deletion in note-based approaches, and at least in some cases this seems to be a major source of errors. In the melodic-contour approach (Mazzoni and Dannenberg 2001), time is divided into equal-length frames, and the fundamental frequency of the query is estimated in each frame. Similarly, the target melody is split into equal-length frames, ignoring note boundaries. When frame boundaries do not line up with note boundaries, we use the note that is most contained in the time frame. Also, because the exact timing of note endings does not seem to be critical or consistent, we extend all notes until the next onset, eliminating rests.

We use two different techniques to deal with tempo variation. First, we time-scale the target data using scale factors of 0.5, 1.0, and 2.0. (Finer-grained scaling does not seem to help.) This ensures that the query and the target will be at least roughly the same tempo for one of the scale factors. Then, dynamic time-warping (DTW) is used to find a good alignment of the query to the target. Dynamic time warping effectively inserts or removes frames to achieve a better match between two sequences of frames.

Ordinarily, DTW allows rather drastic shifts in time and tempo to align two contours. With these unrestricted jumps in time, it is often possible to match two dissimilar musical contours, leading to incorrect melody retrieval. The DTW calculation pattern we use is shown in Figure 4. This pattern prevents consecutive skips or insertions so that local tempo changes are limited to a factor of two. This was the best pattern among several we tried for estimating melodic similarity (Hu and Dannenberg 2002).

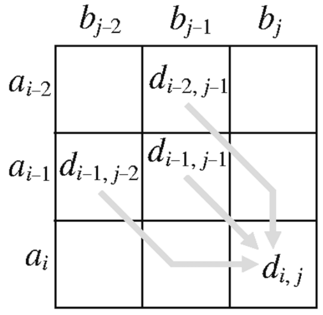
Queries rarely match an entire target in the database, so it is important not to penalize a good match to some subsequence of a target. Rather than perform DTW on every subsequence, which would be very expensive, we can organize the DTW such that the cost of skipping any prefix and any suffix of the target is zero. Transposition is handled by folding all pitches into one octave and running each search with 24 different quarter-step transpositions. The primary difference between this matcher and the Note-Interval matcher is that this one aligns equal-duration frames rather than notes. Furthermore, the contour representation is not invariant to transposition or tempo change, so we must search over many different transpositions and time scales.

Hidden Markov-Model Matching

When auditing a sung query—or indeed any musical production—a trained ear can recognize certain problems: pitch drift, intonation problems, rhythm errors, tempo fluctuations, and so forth. It is quite common for a music teacher to comment to a student that ''the third note was flat,'' or ''you are speeding up in the third measure.'' These two statements represent two fundamentally different views of error: the first indicates a belief that a single note was ''off,'' and the second indicates a belief that a trend is occurring. We refer to these categories of error as ''local'' and ''cumulative,'' respectively, reflecting the scope of the error's effects.

Johnny Can't Sing (JCS) (Meek and Birmingham 2002a) is a system supporting the simultaneous modeling of local and cumulative error in pitch and rhythm. This system provides a unique opportunity to examine the relative significance of these two

Figure 4. *The calculation for the dynamic time warping in the Melodic-Contour search.*



$$d_{i,j} = \mathbf{min} \begin{Bmatrix} d_{i-1,j-1} \\ d_{i-2,j-1} + w(a_{i-1}, b_j) \\ d_{i-1,j-2} + w(a_i, b_{j-1}) \end{Bmatrix} + w(a_i, b_j),$$
$$(1 \le i \le m,\ 1 \le j \le n)$$

categories. A detailed description of the training and matching algorithms used by JCS can be found in a technical report (Meek and Birmingham 2002b).

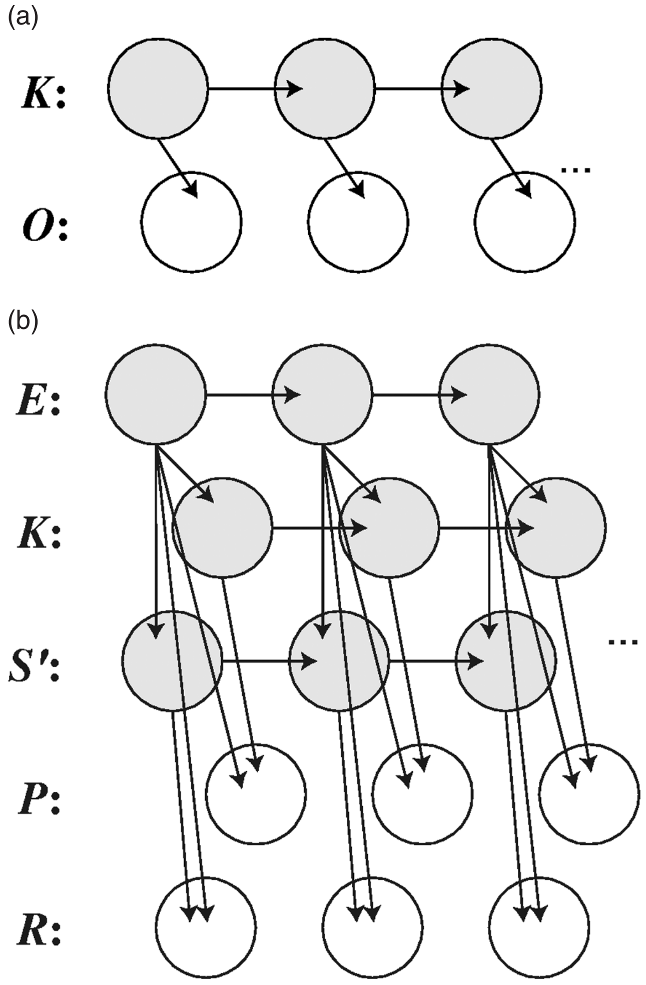
JCS is an extended Hidden Markov Model (HMM) (Rabiner 1989) that associates the notes in a query with the notes in a target through a sequence of hidden states. The fundamental errors (transposition and tempo difference) recommend a fairly detailed state definition to describe this relationship. Each alignment of target and query notes must be considered in each of the possible tempo and transposition contexts. Consider, for instance, the octave-invariant pitch class representation: there are twelve possible transpositions given semitone quantization. Furthermore, we must model tempo differences. Consider a rhythm quantization scheme that allows for nine tempo mappings. In a song with n notes, there are thus $12 \cdot 9 \cdot n$ states, ignoring the various alignment or edit permutations.

In Figure 5a, the conventional HMM dependency structure is shown. The hidden states S are each defined by a tuple, $s_i = \langle E[i], K[i], S'[i]\rangle$, and according to the first-order Markov assumption, the current state depends only on the previous state. $E[i]$ is the "Edit" type associated with the state, defining the way in which query and target notes align. $K[i]$ is the "Key" component, or the transposition relating the pitch in the target to the pitch in the query. $S'[i]$ is the "Speed," or the tempo mapping in the transformation.

Observations O are assumed to depend only on the hidden state and are defined by $o_t = \langle Pitch, Rhythm\rangle = \langle P[t], R[t]\rangle$. Given this view of the query world, we need to determine—using machine-learning techniques or by arduous hand-labeling—the probability of each combination of pitch and rhythm in the query observation given each combination of alignment, transposition, and tempo in the hidden state. It quickly becomes infeasible to explicitly model each of these states. Distributed state representations help control this complexity. The idea is to assume some degree of independence between the components of a model. The second view isolates the components of a hidden state and the components of an observation (see Figure 5b), and it illustrates a more reasonable interpretation of the dependencies among these components. Only the previous edit information E determines the likelihood of various legal extensions to the alignment. The transposition K depends on both the previous transposition and the current edit type, because the degree of modulation and the current position in the target influence the probability of arriving at some transposition level. A pitch observation P depends only on the current edit type and the current transposition, which tell us which pitch we expect to observe. The "emission" probability is then simply the probability of the resulting error, or discrepancy between what we expect and what we see. There is a similar relationship between the edit type E, tempo S', and rhythm observation R.

A simple example illustrates the musical meaning of these elements. Consider the state of the model where E relates the joining of the first two target notes to a query note, K is a transposition of $+2$ semitones, and S' is a tempo scaling of 1.25. The sequence of transformations corresponding to these components of state is shown in Figure 6, starting from the original target notes. The resulting transformed event is compared with the query

Figure 5. Possible dependency schema for basic and distributed state representations. Shaded circles indicate "hidden" states, and white circles indicate fully observable states. Arrows indicate probabilistic dependencies in the models, which evolve over time from left to right. (a) conventional HMM structure; (b) alternative distributed state structure.



event (shown in black), which is said to have a pitch error of +1 and a rhythm error, expressed as a factor, of 0.8.

Comparing the Search Systems

We conducted tests by recording queries and collecting MIDI files for our database. We wanted to make our tests as realistic as possible, so we gave no special instructions to singers (such as singing "ta ta ta" to simplify note segmentation), and all targets are fully polyphonic MIDI files that are automatically processed to extract themes. The performance of our various algorithms on these tests is not as good as the performance cited for many systems in the literature, so, just for the sake of comparison, we also constructed some easier tests. The ability to manipulate outcomes by choosing experimental conditions is a very important observation. It follows that absolute performance numbers have little meaning, but relative performance can help us to assess different search algorithms.

All of our algorithms return an ordered list of targets, from best match to worst. The rank of the correct answer within the list is also computed. To summarize performance, we count the percentage of answers at rank = 1, rank ≤ 2, and rank ≤ 3. We also compute the mean reciprocal rank (MRR). The MRR is the average value of 1/rank, a value in the range 0 to 1, with higher numbers indicating better performance. To simplify reporting, we scale the MRR to the range 0 to 100.

Figure 6. Interpretation of a JCS state.

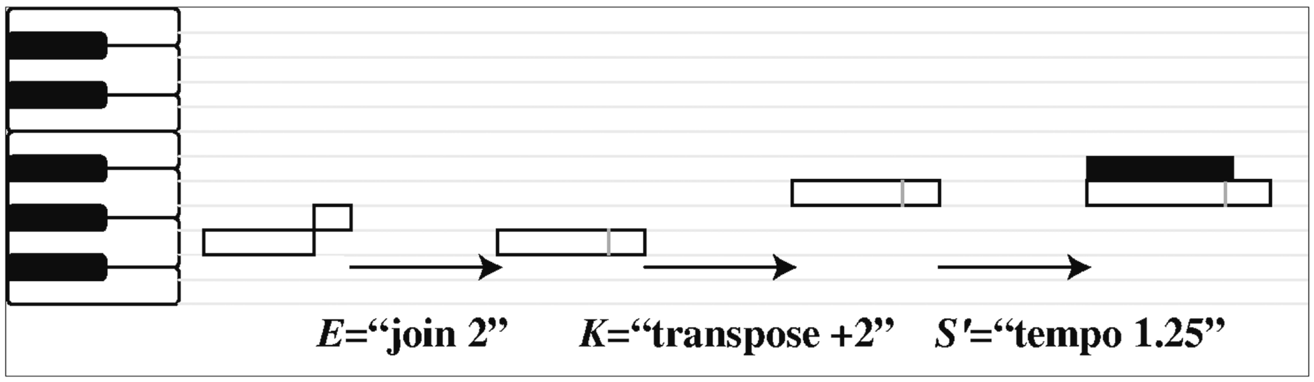
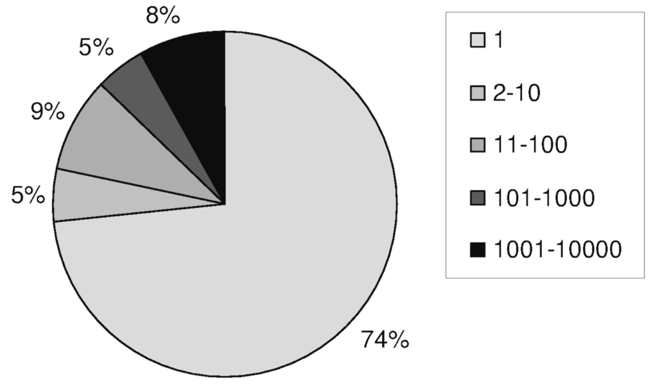


Figure 7. Distribution of ranks for the HMM search algorithm on "high-quality queries."



"High-Quality" Queries

The first set of queries is relatively high in quality, meaning that the queries follow the melody and rhythm of the target song and the recordings are of good quality (i.e., no dropouts or extraneous noise). We have found in previous studies that our algorithms perform quite well when the queries are high-quality. Five individuals, none with vocal training, sang controlled excerpts from eight well-known folk songs, yielding a database of 160 queries. The HMM search system was tested against a massive database of 10,000 synthetically generated targets with a mean length of 40 notes plus the ten folk-song targets used in the queries. The singers, for the most part, were familiar with the folk songs, and sang only contiguous portions of those songs. Using the full HMM model, 59 out of 80 queries returned correct targets ranked first, with an MRR value of 76 (the other 80 were used for training). The distribution of ranks is shown in Figure 7. For the remaining data sets, JCS was used with default parameters and with no training.

The point of this test is to establish that good performance can be obtained under favorable conditions, namely when queries are fairly in-tune subsequences of the targets. In the next section, we will see that performance is highly dependent upon queries and databases. This is one of the reasons that our testbed is so important for our research.

"Ordinary" Queries

We have two more collections of queries that turn out to be more difficult than the folk-song queries. Query Set 1 was collected from 10 subjects with no vocal training who were presented 10 Beatles songs. After hearing a song once, each singer was asked to sing the "most memorable" portion of the piece. No instructions were given as to whether they should sing lyrics, and subjects varied in this respect. Subjects were free to try again if they felt their first attempt was bad for some reason. In many cases, subjects made more than one attempt, so there are 131 queries in all. Although most of the queries are recognizable, many of them do not correspond very well to the actual songs (as judged by the authors listening to the queries). Subjects often skipped from one section to another, creating melodic sequences that do not exist in the actual song. It is interesting to note that these fabricated sequences are often completely convincing and do not seem to confuse human listeners. Many singers have mild to severe intonation problems and many added expressive pitch bends to their singing, which complicates note identification. Some queries contain noise caused by touching the microphone, and some contain bits of self-conscious laughter and other sounds.

Query Set 2 was collected from a larger number of subjects. As a class project, students were recruited to record 10 queries each from volunteers, resulting in a collection of 165 usable queries. These were all sung from memory and suffer from many of the same problems as Query Set 1.

The ThemeExtractor program extracted approximately 11 short "themes" from each target song in the database. In all of our systems, searching is performed by comparing the query to each theme from a song. The similarity rating of the best match is reported as the similarity rating of the song. These ratings are then sorted to compute the rank order of the correct song.

Table 1 shows the results of running Query Set 1 against a collection of 258 Beatles songs, for which there are a total of 2,844 themes. It can be seen that the matchers are significantly different in terms of search quality. At least with these queries, it seems that better melodic similarity and error models give better search performance.

Figure 8. Distribution of query problems. We judged only about half the queries to directly correspond to the correct target.

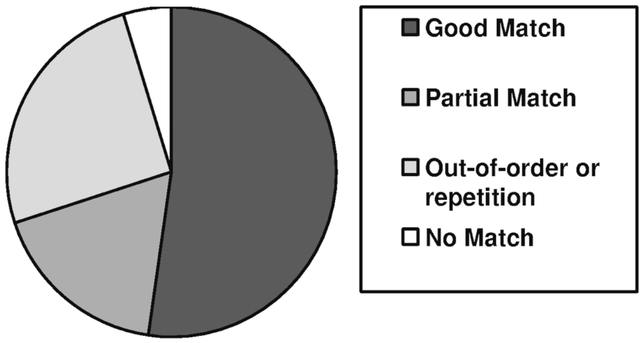


Table 1. Percentage of correct targets returned at or below ranks 1, 2, and 3, and Mean Reciprocal Rank (MRR) for Query Set 1

Search Algorithm	*= 1*	*≤ 2*	*≤ 3*	*MRR*
Note-Interval	8.4	12.2	13.0	13.4
Melodic-Contour	15.3	19.1	21.4	21.0
Hidden Markov Model	20.6	26.7	29.0	27.0

MRR is reported on a scale from 0 to 100

Table 2. Percentage of correct targets returned at or below ranks 1, 2, and 3, and Mean Reciprocal Rank (MRR) for Query Set 2

Search Algorithm	*= 1*	*≤ 2*	*≤ 3*	*MRR*
Note-Interval	21.3	27.1	31.6	28.2
Melodic-Contour	27.7	32.3	32.9	32.9
Hidden Markov Model	25.8	30.3	32.9	31.0

Table 2 shows the results of running Query Set 2 against a collection of 868 popular songs. The total number of themes in this database is 8,926. All three algorithms performed better on these data than with Query Set 1, even though there are many more themes. Unlike in Table 1, where the algorithms seem to be significantly different, all three algorithms in this test have similar performance, with an MRR of about 30. The Note-Interval algorithm is about 100 times faster than the other two, so at least in this test, it seems to be the best, even if its MRR is slightly lower.

The fact that the Note-Interval algorithm works well in this test deserves some comment. In previous work, we compared note-by-note matchers to contour- or frame-based matchers and concluded that the melodic-contour approach was significantly better in terms of precision and recall (Mazzoni and Dannenberg 2001). For that work, we experimented with various note-matching algorithms, but we did not find one that performs as well as the contour matcher. Apparently, the note-matching approach is sensitive to the relative weights given to duration versus pitch, and matching scores are also sensitive to the assigned edit penalties. Perhaps also this set of queries favors matchers that use local information (intervals and ratios) over those that use more global information (entire contours).

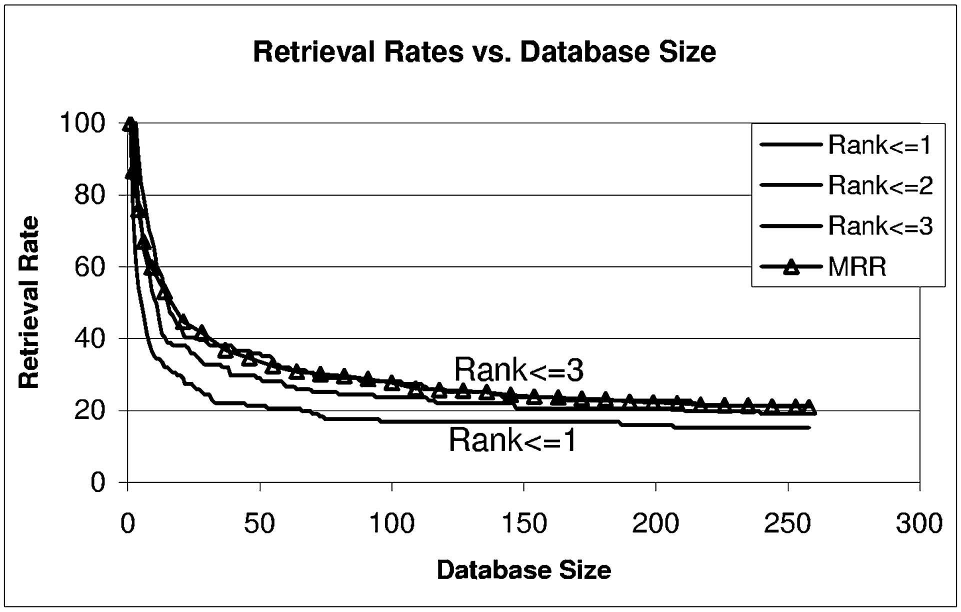
Sources of Error

We have studied where errors arise in these search algorithms. As mentioned, the major problem is that many melodies presented in the queries are simply not present in the original songs. In Set 1, only about half were judged to match the correct target in the database in the sense that the notes of the melody and the notes of the target corresponded. (See Figure 8.) About a fifth of the queries partially matched a target, and a few did not match at all. Interestingly, about one-fourth of the queries matched material in the correct target, but the query contained extra repetitions or out-of-order phrases. An example of this is where subjects alternately hum a melody and a countermelody, even when these do not appear as any single voice in the original song. Another example is where subjects sing two phrases in succession that did not occur that way in the original song. Sometimes subjects repeat phrases that were not repeated in the original. Ultimately, Query-by-Humming assumes reasonably good queries, and more work is needed to help the average user create better queries.

Scaling to Larger Databases

Our experimental algorithms are computationally demanding, so we have limited our studies to medium-sized databases. The Beatles database used with Query Set 1 has 2,844 themes extracted from 258 songs. The database used with Query Set 2 has

Figure 9. Number of targets ranked 1 (bottom curve), 2 or less, 3 or less (top curve), and MRR (triangles) as a function of database size.



8,926 themes extracted from 868 songs. Themes have an average of about 41 notes.

Regardless of the algorithm, an interesting question is always this: How do the results scale as the database grows larger? One way to explore this question is to use the similarity scores to simulate databases of different sizes without actually rerunning the search.

Let us assume we have a table of melodic distance scores for Q queries and T targets, and $S(q,t)$ (where $0 \leq q < Q$, and $0 \leq t < T$) is the distance of the best match of query q to target t. We also have a list of correct targets $C(q)$ for each query. Now, suppose we want to simulate a database of size $N < T$ for some query q. We construct a "random" database by inserting the correct target $C(q)$ and $N-1$ random choices from the set $\{0...T-1\} - \{C(q)\}$. We can compute the rank of the correct target in such a random database **R** by counting how many entries in the database have a lower score than the score for the correct target:

$$\text{rank} = 1 + \left|\{x\colon x \in \mathbf{R} \text{ and } S(q, x) < S(q, C(q))\}\right|$$

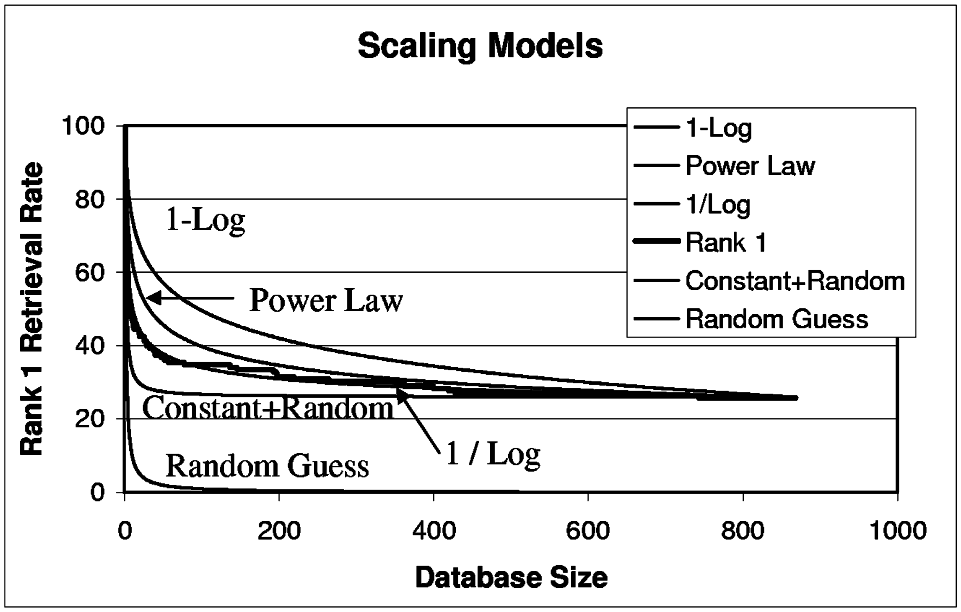
This gives us the rank for a particular random database **R**, and we would need to run this simulation many times to estimate the expected rank.

In practice, we want to consider all queries (not just the single query q) and we want results for all sizes of databases in order to study the trend. To accomplish this, we "grow" the random database for each query. Initially, each query's database has only the correct target. Then we grow each database by one target selected randomly from the targets not yet included. Each time we grow the database, we compute the number of correct targets at rank 1, rank 2, etc. These numbers can then be plotted as a function of database size as in Figure 9, which is based on Query Set 1 and the Melodic-Contour search.

Note that the function becomes flat, indicating that the number of correct answers does not fall off rapidly as the database size increases. Various functions could be used to approximate and extrapolate the observed data. Figure 10 shows the Rank 1 curve together with various candidate models of search performance as a function of database size.

The simplest model is that the search procedure simply returns a random guess. The expected number of correct results at rank 1 is $y = Q/N$, where y is the number of correct targets returned with rank 1, Q is the number of queries, and N is the database size. This rapidly converges to zero and is a

Figure 10. Various models of database scaling with observed data for Rank 1.



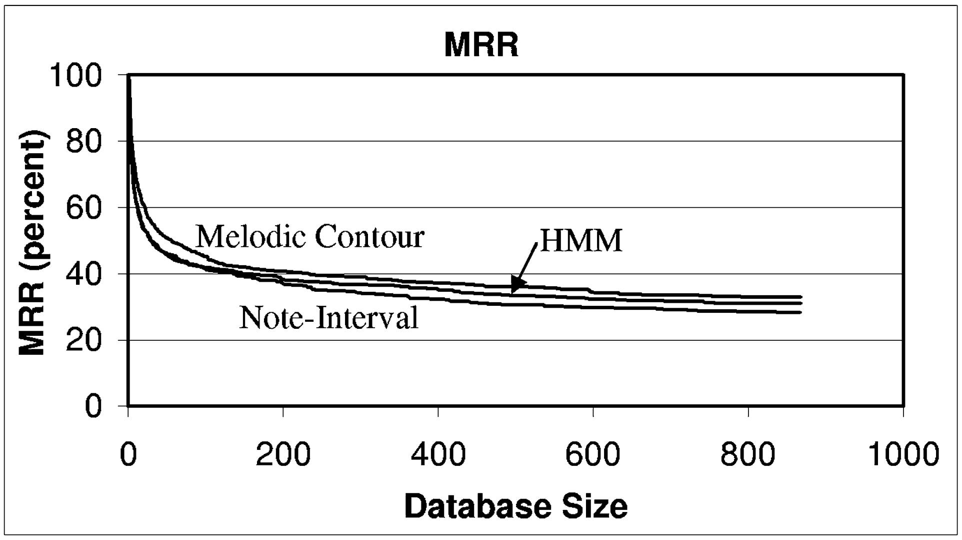
poor fit to the data (as one would hope!). A slight modification to this has a set of queries where the search is perfect, regardless of database size, combined with another set of queries where the search is ineffective and returns a random guess. The corresponding equation is $y = c + (Q - c)/N$, for some constant c. Note that in this model, the searches that really "work" are independent of the database size. This model, labeled "Constant + Random" in Figure 10, converges more rapidly to the constant c than do the observed data.

Another possible model is a power-law model, $y = N^{-p}$. The corresponding curve (labeled "Power Law") is not as "flat" as the observed data. A function that flattens quickly is the logarithm, so we tried two forms based on log(N). The equation $y = Q(1 - c \cdot \log(N))$ does not conform to the observations (see the "1 – Log" curve), but the equation $y = c_1/\log(c_2 \cdot N)$ fits the data reasonably well, albeit with two parameters (see the "1/Log" curve). Of course, there is no proof that we can extrapolate this function to predict behavior with larger databases, but it is encouraging that the function decreases slowly. For example, to reduce the number of correct results at Rank 1 in this model by half, the database size must be squared.

Figure 11 is similar to Figure 9, but it plots the MRR from all three search systems using Query Set 2. These data seem to confirm the general 1/log(N) scaling trend. At least in our limited examples, the scaling trend seems to be independent of the set of queries, the database, and the search algorithm.

These data shows that mean rank is not a good measure of performance. For example, a ranker that returns the correct answer ranked first half of the time and ranked 100th half of the time is a better performer than one that returns the correct answer randomly between first and 100th, even though their mean rank is the same. Figure 12 shows a histogram of ranks returned from Query Set 1. There is some significant fraction of "correctly matched" results with very low ranks (see Figure 12a), but the rest (queries for which no good match was found) are almost randomly distributed (see Figure 12b). The mean rank is the "center of gravity" of this histogram, and it will obviously grow with the database size. On the other hand, the number of low-ranking correct targets will remain nearly constant as shown in Figure 11. In these tests, the MRR seems to be highly correlated with the proportion of correct answers ranked in the top two or three.

Figure 11. MRR as a function of database size for three different search algorithms. All three follow the same general 1/Log trend.



Summary and Conclusions

It is widely understood and agreed that better evaluation tools are needed in the field of Music-Information Retrieval. We have constructed a Query-by-Humming testbed to evaluate and compare different search techniques. The testbed helps us to organize experiments by providing explicit representations and standard formats for queries, targets, collections (subsets of queries or targets), preprocessing stages, search algorithms, and result reporting. A single command can run a complete test, including the preprocessing of data, searching for a set of queries, and generating reports. Most of our testbed, including some of the databases, is available to other researchers, and we can also collaborate with other researchers by adding new search systems into our testbed. Please contact the authors for information about formats and APIs.

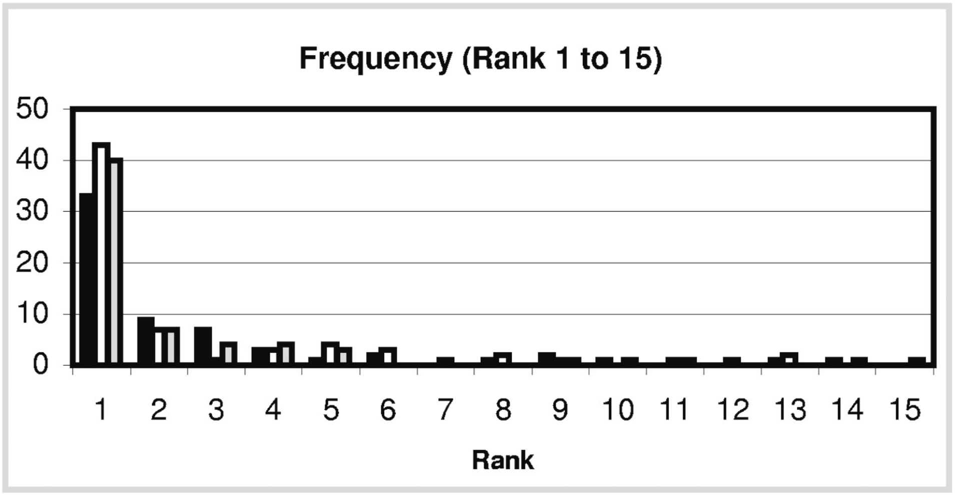
We have compared three algorithms for music search that have been reported previously, but we have never compared them in a "head-to-head" fashion. The Note-Interval algorithm treats music as sequences of pitch intervals and IOI ratios and searches for an alignment that minimizes a distance function. The Contour-Matching algorithm, a variation of string matching, does not segment the query into notes, but uses dynamic time-warping to find the best match to melodic contour. The HMM approach matches notes using a probabilistic error model intended to account for the kinds of errors observed in queries.

The Contour-Matching and HMM algorithms are extremely slow, taking on the order of two seconds of computation time per entry in the database, which translates into days of runtime for many of our tests. Although this may be impractical for many tasks, we believe it is important to discover the best search techniques possible in terms of precision and recall. Until quite recently, these algorithms seemed to outperform all faster approaches. However, at least on Query Set 2, our current Note-Interval system delivers similar search quality with a run time of about 0.02 sec per entry in the database, making it the clear winner in our comparison. Interestingly, the HMM and contour-matching approaches do not make the same mistakes, so returning the top choice of each is superior to returning the top two choices of either algorithm.

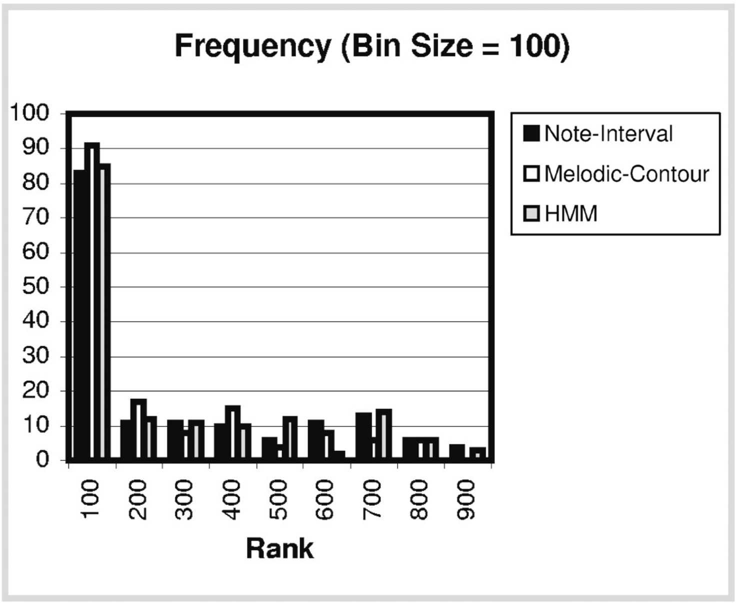
Our work shows a wide range of performance according to the quality of queries. When queries contain a reasonably long sequence of well-sung pitches, search algorithms can be very effective. On the other hand, when we collected queries from general university populations, we found many queries that were very difficult to match. The wide range in performance of our systems on different query sets should serve as a warning to researchers: performance is highly dependent on queries, so no

Figure 12. Histograms showing how correct targets are ranked for each of the three systems using Query Set 2. The first histogram (a) shows details of the first 15 ranks. The second histogram (b) includes all of the data, but with large bins of 100 ranks each.

(a)



(b)



comparison is possible without controlling the query set.

Finally, we propose that the issue of scaling with database size can be studied by simulation. Given distance or similarity estimates between queries and targets, we can plot the expected number of queries whose correct targets will be ranked 1 (or in general, less than some rank k). For our algorithms, we found that a $1/\log(N)$ model gives a reasonable fit to the observed data. This is encouraging, because this function becomes very flat as the database size increases.

Acknowledgments

We gratefully acknowledge the support of the National Science Foundation under grant IIS-0085945.

The opinions in this paper are solely those of the authors and do not necessarily reflect the opinions of the funding agency. This work was done while William P. Birmingham and Colin Meek were at the University of Michigan and George Tzanetakis was at Carnegie Mellon University. Thanks to Dominic Mazzoni and Mark Bartsch for initial implementation of some testbed components. Thanks to Chee Kiat, Crystal Fong, and David Murray for help with data collection and analysis. Finally, thanks to the organizers and reviewers at ISMIR 2003, where an earlier version of this article (Dannenberg et al. 2003) was presented.

References

Bainbridge, D., M. Dewsnip, and I. H. Witten. 2002. ''Searching Digital Music Libraries.'' In *Digital Libraries: People, Knowledge, and Technology: 5th International Conference on Asian Digital Libraries*. New York: Springer-Verlag, pp. 129–140.

Birmingham, W. P., et al. 2001. ''MUSART: Music Retrieval Via Aural Queries.'' In *2nd Annual International Symposium on Music Information Retrieval*. Bloomington, Indiana: Indiana University, pp. 73–81.

Dannenberg, R. B., et al. 2003. ''The MUSART Testbed for Query-by-Humming Evaluation.'' In H. H. Hoos and D. Bainbridge, eds. *ISMIR 2003: Proceedings of the Fourth International Conference on Music Information Retrieval*. Baltimore, Maryland: Johns Hopkins University, pp. 41–50.

Downie, J. S. 2002. ''Panel in Music Information Retrieval Evaluation Frameworks.'' In *ISMIR 2002 Conference Proceedings*. Paris: IRCAM, pp. 303–304.

Durbin, R., et al. 1998. *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids*, Cambridge, UK: Cambridge University Press, pp. 22–24.

Futrelle, J., and J. S. Downie. 2002. ''Interdisciplinary Communities and Research Issues in Music Information Retrieval.'' In *ISMIR 2002 Conference Proceedings*. Paris: IRCAM, pp. 215–221.

Ghias, A., et al. 1995. ''Query by Humming: Musical Information Retrieval in an Audio Database.'' In *Proceedings of the Third ACM International Conference on Multimedia*. New York: ACM Press, pp. 231–236.

Hu, N., and R. B. Dannenberg. 2002. ''A Comparison of Melodic Database Retrieval Techniques Using Sung Queries.'' In *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital libraries*. New York: ACM Press, pp. 301–307.

Mazzoni, D., and R. B. Dannenberg. 2001. ''Melody Matching Directly From Audio.'' In *2nd Annual International Symposium on Music Information Retrieval*. Bloomington, Indiana: Indiana University, pp. 17–18.

McNab, R. J., et al. 1996. ''Towards the Digital Music Library: Tune Retrieval from Acoustic Input.'' In *Proceedings of the first ACM international conference on Digital Libraries*. New York: ACM Press, pp. 11–18.

Meek, C., and W. P. Birmingham. 2001. ''Thematic Extractor.'' In *2nd Annual International Symposium on Music Information Retrieval*. Bloomington, Indiana: Indiana University, pp. 119–128.

Meek, C., and W. P. Birmingham. 2002a. ''Johnny Can't Sing: A Comprehensive Error Model for Sung Music Queries.'' In *ISMIR 2002 Conference Proceedings*. Paris: IRCAM, pp. 124–132.

Meek, C., and W. P. Birmingham. 2002b. *Johnny Can't Sing: A Comprehensive Error Model for Sung Music Queries.* Technical Report CSE-TR-471-02, University of Michigan, Ann Arbor.

Pardo, B., and W. P. Birmingham. 2002. ''Encoding Timing Information for Musical Query Matching.'' In *ISMIR 2002 Conference Proceedings*. Paris: IRCAM, pp. 267–268.

Pardo, B., W. P. Birmingham, and J. Shifrin. 2004. ''Name That Tune: A Pilot Study in Finding a Melody from a Sung Query.'' *Journal of the American Society for Information Science and Technology* 55(4):283–300.

Pauws., S. 2002. ''CubyHum: A Fully Operational Query-by-Humming System.'' In *ISMIR 2002 Conference Proceedings*. Paris: IRCAM, pp. 187–196.

Rabiner, L. 1989. ''A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.'' *Proceedings of IEEE* 77(2):257–286.

Roads, C. 1996. *The Computer Music Tutorial*. Cambridge, Massachusetts: MIT Press, pp. 509–511.

Shifrin, J., et al. 2002. ''HMM-Based Musical Query Retrieval.'' In *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries*. New York: ACM Press, pp. 295–300.

Tolonen, T., and M. Karjalainen. 2000. ''A Computationally Efficient Multi-Pitch Analysis Model.'' *IEEE Transactions on Speech and Audio Processing* 8(6): 708–716.

Zeidler, S. 2003. ''U.S. CD Sales Turn Up.'' Available online at www.forbes.com/newswire/2003/11/17/rtr1150606.html.