The MUSART Testbed for Query-By-Humming Evaluation [*]

Roger B. Dannenberg, William P. Birmingham, George Tzanetakis, Colin Meek, Ning Hu, Bryan Pardo

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1-412-268-3827
rbd@cs.cmu.edu

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2110
+1-734-936-1590
wpb@eecs.umich.edu

*Published in: *Proceedings of the Fourth International Conference on Music Information Retrieval,* Baltimore, Maryland, USA, October 2003. Baltimore: Johns Hopkins Univeristy, 2003. pp. 41-50.

Abstract

Evaluating music information retrieval systems is acknowledged to be a difficult problem. We have created a database and a software testbed for the systematic evaluation of various query-by-humming (QBH) search systems. As might be expected, different queries and different databases lead to wide variations in observed search precision. "Natural" queries from two sources led to lower performance than that typically reported in the QBH literature. These results point out the importance of careful measurement and objective comparisons to study retrieval algorithms. This study compares search algorithms based on note-interval matching with dynamic programming, fixed-frame melodic contour matching with dynamic time warping, and a hidden Markov model. An examination of scaling trends is encouraging: precision falls off very slowly as the database size increases. This trend is simple to compute and could be useful to predict performance on larger databases.

1 Introduction

The MUSART project is a collaboration between the University of Michigan and Carnegie Mellon University. Together, we have been exploring the design of query-by-humming systems (Birmingham, et al., 2001; Hu & Dannenberg, 2002; Meek & Birmingham, 2002; Pardo & Birmingham, 2002; Shifrin, et al., 2002). We have developed a variety of algorithms based on Markov models, 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 & Downie, 2002) Many results are 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. We have integrated several of our research systems into this 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.

In the next section, we describe the architecture of our testbed. Then, in Section 3, we describe three different search systems we have studied. In Section 4 we present some results of our algorithm comparisons. Section 5 discusses the general sources of error we observed. In Section 6, we discuss the issue of search performance as we scale to larger databases. Section 7 presents some discussion and conclusions.

2 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 that complete tests should run from start to finish without manual intervention. A typical test starts 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 (currently in Pennsylvania, Michigan, and Washington) can have convenient access to all results from all tests.

In order 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.

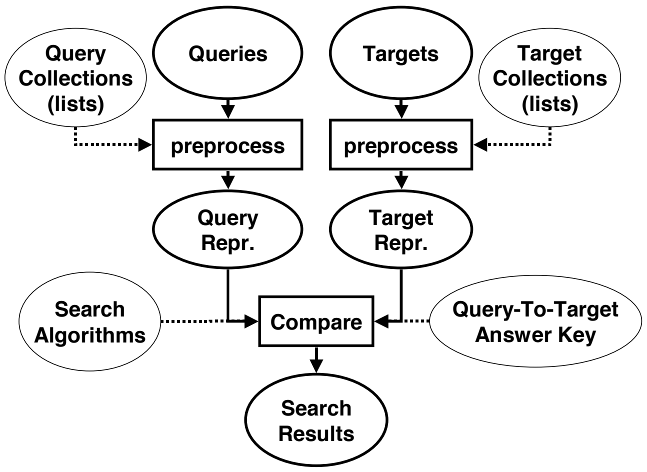


Figure 1: Architecture of the Musart Testbed.

In addition to queries and targets, we have intermediate representations. For example, most search systems convert queries to transcriptions stored as MIDI files or to pitch contours stored as data in text files. We usually use the "Thematic Extractor" to obtain themes from target MIDI files and search the themes rather than the full MIDI target file. 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. We keep a file for each query that lists all the correct targets, since there may be several versions of a song in the database, appearing as several different 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 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:

- 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
- For each query: the match score for each target and the rank order of the correct target.
- Statistics: 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.

3 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 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 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.

3.1 Note-Interval, Dynamic Programming Search

The Note-Interval system relies on a query transcriber to estimate note onset times and pitches in audio queries. Both targets and queries are then transformed into sequences of note-intervals, each of which consists of a pitch interval and a rhythmic interval. Pitch intervals are quantized to the half-step and range from –12 to +12 half steps. Rhythmic intervals are represented as one of five log-spaced Inter Onset Interval Ratio (IOIr) values (Pardo & Birmingham, 2002). This encoding is both tempo-invariant and transposition-invariant. Once encoded, targets are ranked by similarity to the query. Similarity is given by the minimum cost of transforming a 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 & Birmingham, 2002; Pardo, Birmingham, & Shifrin, 2003) 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 decreases exponentially with distance in either IOIr or pitch-interval.

3.2 Melodic-Contour, Dynamic Time Warping Search

The melodic-contour matcher is based on the idea that while pitch estimation is not too difficult, 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 & 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. Dynamic time-warping is used to find a good alignment of the query to the target. 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.

3.3 Hidden Markov Model Matching

Johnny Can't Sing (JCS) (Meek & Birmingham, 2002a; Meek & Birmingham, 2002b) is a hidden Markov model matcher that uses a distributed state representation to model both "cumulative" and "local" error. This means that, like the note-interval approach, JCS explicitly models changes in tempo and pitch-center, and like the melodic-contour approach, models errors that have a purely local effect on the pitch and rhythm of the query. A note-based approach, we incorporate the notion of fragmentation and consolidation (Mongeau & Sankoff, 1990), but the state model also supports arbitrary gaps in the query and target with low probability.

4 Results of Comparisons

We have conducted tests with different sets of queries and databases. In all of the work reported here, there were no special instructions for singers (such as singing "ta ta ta") and all targets are fully polyphonic MIDI files which are automatically processed to extract themes. 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 drop outs or extraneous noise). We have found in previous studies that our algorithms perform quite well when the queries are high quality. We also collected new, larger sets of queries of lower quality, and found that, with these, the search performance of all algorithms was much worse. Below, we compare these sets of queries. We then compare our three 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, or rank ≤ 3. We also compute the MRR (mean reciprocal rank). 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.

4.1 The "High-Quality" Queries

Five queriers, two musically trained, sang controlled excerpts from ten 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) in order to test scalability, given queries collected in ideal circumstances. The singers were – for the most part – familiar with the folk songs, and sang only contiguous portions of those songs. Using the full HMM model, 59 out of 80 queries (the other 80 were used for training) returned correct targets ranked first, with an MRR value of 76. The distribution of ranks is shown in Figure 2. For the remaining data sets, JCS is used with default parameters, with no training.

The point of this test is to establish that good performance can be obtained under reasonable conditions, namely that queries are fairly in-tune sub-sequences 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.

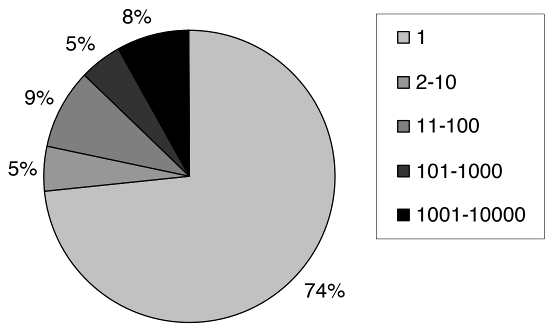


Figure 2: Distribution of ranks for the HMM search algorithm on "high quality queries".

4.2 The "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. While 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 are all sung from memory and suffer from many of the same problems as Query Set 1.

A preprocessing step (Meek & Birmingham, 2001) extracts approximately 11 short "themes" from each target song in the database. In all of our systems, search 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 2844 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.

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 8926. All three algorithms performed better on this 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.

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

Table 1: Percentage of correct targets returned at or below ranks 1, 2 and 3, and Mean Reciprocal Rank (MRR) for Query Set 1. MRR is reported on a scale from 0 to 100.

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

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 & 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).

5 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 were in correspondence. (See Figure 3.) 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 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.

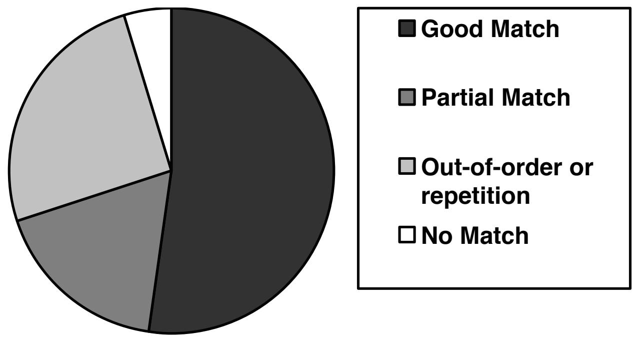


Figure 3: Distribution of query problems. We judged only about half the queries to have a direct correspondence to the correct target.

6 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 2844 themes extracted from 258 songs. The database used with Query Set 2 has 8926 themes extracted from 868 songs. Themes have an average of about 41 notes.

Regardless of the algorithm, an interesting question is always: 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 re-running the search.

Let us assume we have a table of melodic distance scores for Q queries and T targets: $S(q,t)$ (where $0 \le q < Q$, and $0 \le t < T$) 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 $\{0\ldots T-1\}-\{C(q)\}$. 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:

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

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 4, which is based on Query Set 1 and the Melodic-Contour search.

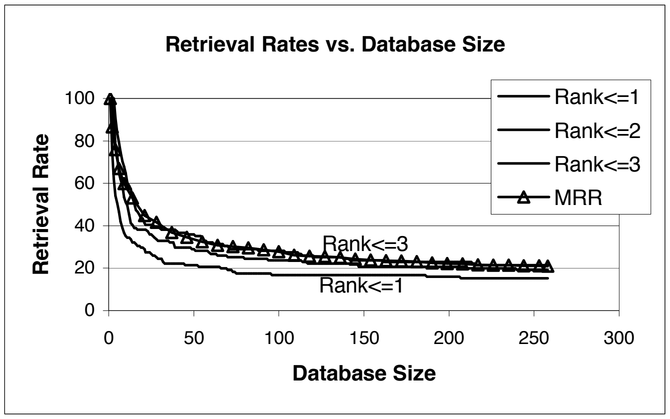


Figure 4: Number of targets ranked 1 (bottom curve), 2 or less, 3 or less (top curve), and MRR (triangles) as a function of database size.

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 5 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 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 5 converges more rapidly to the constant c than does 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 the 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*.

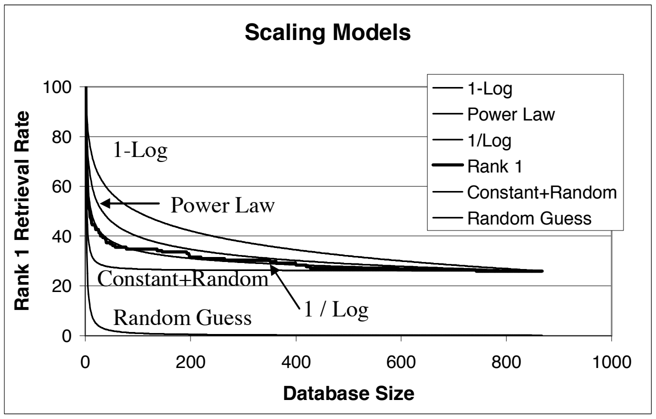


Figure 5: Various models of database scaling with observed data for Rank 1.

Figure 6 is similar to Figure 4, but it plots the MRR from all three search systems using Query Set 2. This data seems 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.

This data shows that mean rank is not a good measure of performance. For example, a ranker that returns the correct answer ranked 1st half the time and ranked 100th half the time is a better performer that one that returns the right answer randomly between 1st and 100th, even though their mean rank is the same. Figure 7 shows a histogram of ranks returned from Query Set 1. There is some significant fraction of "correctly matched" results with very low ranks, but the rest (queries for which no good match was found) are almost randomly distributed. 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 4. In these tests, the MRR seems to be highly correlated with the proportion of correct answers ranked in the top two or three.

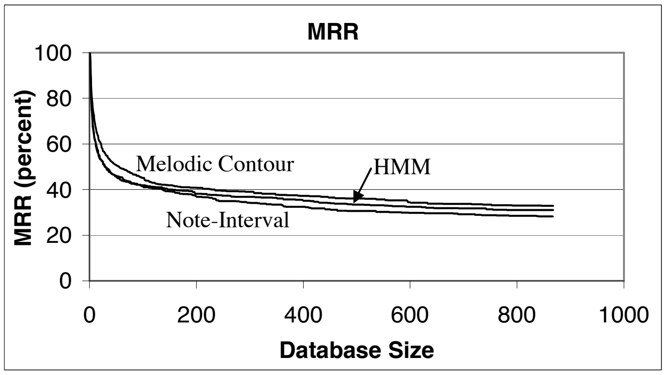
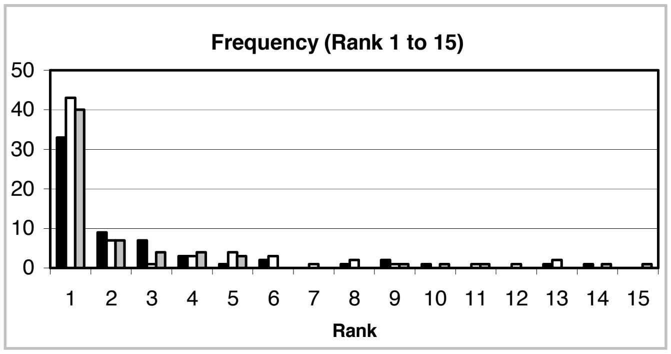


Figure 6: MRR as a function of database size for three different search algorithms. All three follow the same general 1/Log trend.



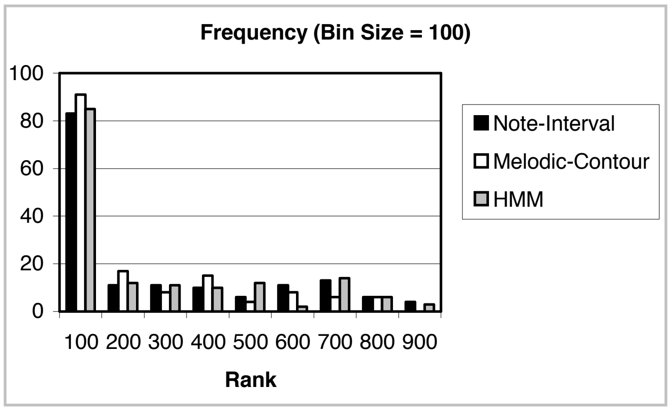


Figure 7: Histograms showing how correct targets are ranked for each of the three systems using Query Set 2. The upper histogram shows details of the first 15 ranks. The lower histogram includes all of the data, but with large bins of 100 ranks each.

7 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 never compared 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 2 seconds of computation time per entry in the database, which translates into days of runtime for many of our tests. While 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 out-perform 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 seconds 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 (which, if anything, might be expected to produce better queries than the overall population), 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 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 flat as the database size increases.

Acknowledgements

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 agencies. 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.

References

Birmingham, W. P., Dannenberg, R. B., Wakefield, G. H., Bartsch, M., Bykowski, D., Mazzoni, D., Meek, C., Mellody, M., & Rand, W. (2001). "MUSART: Music Retrieval Via Aural Queries." *International Symposium on Music Information Retrieval*. pp. 73-81.

Downie, J. S. (2002). "Panel in Music Information Retrieval Evaluation Frameworks." *ISMIR 2002 Conference Proceedings*. IRCAM, pp. 303-304.

Futrelle, J., & Downie, J. S. (2002). "Interdisciplinary Communities and Research Issues in Music Information Retrieval." *ISMIR 2002 Conference Proceedings*. IRCAM, pp. 215-221.

Ghias, A., Logan, J., Chamberlin, D., & Smith, B. C. (1995). "Query by humming - musical information retrieval in an audio database." *Proceedings of ACM Multimedia 95*. pp. 231-236.

Hu, N., & Dannenberg, R. B. (2002). "A Comparison of Melodic Database Retrieval Techniques Using Sung Queries." *Joint Conference on Digital Libraries*. Association for Computing Machinery.

Mazzoni, D., & Dannenberg, R. B. (2001). "Melody Matching Directly From Audio." *2nd Annual International Symposium on Music Information Retrieval*. Bloomington: Indiana University, pp. 17-18.

McNab, R. J., Smith, L. A., Witten, I. H., Henderson, C. L., & Cunningham, S. J. (1996). "Towards the digital music library: Tune retrieval from acoustic input." *Proceedings of Digital Libraries '96*. ACM.

Meek, C., & Birmingham, W. P. (2001). "Thematic Extractor." *2nd Annual International Symposium on Music Information Retrieval*. Bloomington: Indiana University, pp. 119-128.

Meek, C., & Birmingham, W. P. (2002a). "Johnny Can't Sing: A Comprehensive Error Model for Sung Music Queries." *ISMIR 2002 Conference Proceedings*. IRCAM, pp. 124-132.

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

Mongeau, M., & Sankoff, D. (1990). Comparison of Musical Sequences. In W. Hewlett, *et al*. (Eds.), *Melodic Similarity Concepts, Procedures, and Applications* (Vol. 11). Cambridge: MIT Press.

Pardo, B., & Birmingham, W. P. (2002, , October 13-17). "Encoding Timing Information for Musical Query Matching." *ISMIR 2002, 3rd International Conference on Music Information Retrieval*. IRCAM, pp. 267-268.

Pardo, B., & Birmingham, W. P. (2002). "Improved Score Following for Acoustic Performances." *Proceedings of the 2002 International Computer Music Conference*. San Francisco: International Computer Music Association.

Pardo, B., Birmingham, W. P., & Shifrin, J. (2003). "Name that Tune: A Pilot Studying in Finding a Melody from a Sung Query." *Journal of the American Society for Information Science and Technology*, (in review).

Pauws., S. (2002). "CubyHum: A Fully Operational Query-by-Humming System." *ISMIR 2002 Conference Proceedings*. IRCAM, pp. 187-196.

Shifrin, J., Pardo, B., Meek, C., & Birmingham, W. P. (2002). "HMM-Based Musical Query Retrieval." *Joint Conference on Digital Libraries*. Association for Computing Machinery, pp. 295-300.