UNDERSTANDING SEARCH PERFORMANCE IN QUERY-BY-HUMMING SYSTEMS

Roger B. Dannenberg and Ning Hu
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA

ABSTRACT

Previous work in Query-by-Humming systems has left open many questions. Although a variety of techniques have been explored, there has been relatively little work to compare them under controlled conditions, especially with "real" audio queries from human subjects. Previous work comparing note-interval matching, melodic contour matching, and HMM-based matching is extended with comparisons to the Phillips CubyHum algorithm and various n-gram search algorithms. We also explore the sensitivity of note-interval dynamic programming searches to different parameters and consider two-stage searches combining a fast n-gram search with a more precise but slower dynamic programming algorithm.

Keywords: Query-by-Humming, N-gram, Dynamic Programming, Evaluation

1. INTRODUCTION

The MUSART project has studied and compared techniques for query-by-humming (QBH) music retrieval. [1] In this work, audio queries are used to search symbolic (MIDI) targets in a database. Because we use "real" queries from non-musicians, the general quality of queries is low, making retrieval quite challenging. The difficulty of our task and configuration can be seen as a feature: we are far from any ceiling effects, so any significant improvement will show up clearly in the results.

Perhaps the most difficult problem for QBH systems is the determination of melodic similarity. Our previous work considered several algorithms for computing melodic similarity. One is based on the fairly well-known dynamic programming algorithms for string matching, which had already been applied to the melodic similarity problem. [2, 3] We use transposition- and tempo-invariant representations based on pitch intervals and inter-note-onset intervals, so we refer to this as the *note-interval search.* From previous experience, we knew of numerous shortcomings of this approach, and thought we could find better approaches.

One of these we call *melodic contour matching* [4], which represents melody as a continuous function of pitch versus time. The advantage of melodic contour matching is that it does not require note segmentation. Instead, only fundamental frequency estimates are used, eliminating segmentation as a source of errors.

The other more sophisticated approach is a *hidden Markov model* (*HMM*) [5] in which different categories of errors are considered in a unified manner. In contrast to the dynamic programming approach, the hidden Markov model approach can consider explicitly different error types. One type of error is an incorrect pitch. Another is a transposition, where every note from a certain point onward is transposed. Rhythm and tempo errors can also be considered, and different probabilities can be assigned to different error types.

Early experiments with both the melodic contour and HMM approaches showed improvements over string-matching approaches. However, Pardo's work with note interval search [6] resulted in improvements over earlier versions. Ultimately, all three approaches showed roughly equal performance in our MUSART testbed. [7]

The fact that the note interval search was initially unimpressive and later delivered quite good performance led us to investigate the algorithm further to study the sensitivity of different parameter settings. It appears that other researchers may have drawn incorrect conclusions from less-than-optimal implementations

Along the same lines, there might be other variations in the dynamic programming algorithm that could lead to even better performance. The CubyHum system [8] uses a fairly elaborate set of rules to compute melodic similarity. These rules are designed to model various error types, somewhat like the way the MUSART HMM system models errors. We compare the performance of the CubyHum approach to our other algorithms.

Another unanswered question is whether there is any way to avoid a search cost that is linear in the size of the database. All of the algorithms we studied compare the query to each target in the database. It would be much better to build some sort of index to avoid the cost of exhaustive search. A promising approach is to narrow the field of potential targets using an indexing scheme based on n-grams, and then search these candidate themes with a slower, but higher-precision algorithm.

Thus, our intention is to wrap up some "loose ends" from our previous work and answer some nagging questions about QBH systems. After describing some related work, we present experiments with a simplified note-interval search algorithm to determine its sensitivity and optimal settings. In Section 4, we describe our reimplementation and evaluation of CubyHum within the MUSART testbed. In Section 5, we investigate how n-gram-based search performs with difficult vocal queries. There will undoubtedly be more questions and research,

but we feel that we now have a fairly good picture of a wide range of options and the potential performance that can be obtained in QBH systems using realistic queries.

2. RELATED WORK

Our study of note-interval search uses a simplified version of the note-interval algorithm developed by Pardo [6], and our findings are consistent with his. While Pardo emphasizes his optimization process and overall results, we are more interested in sensitivity to varying parameter values. We found that search results are highly dependent upon the quality of queries. Lesaffre, et al. [9] collected and studied vocal queries.

N-grams are studied extensively in the text-retrieval community and have also found application in music information retrieval. Downie [10, 11] evaluated n-gram techniques and explored the effects of different design parameters. Errors were simulated by changing one pitch interval randomly. Uitenbogerd and Zobel [12] considered 4-grams of different pitch interval representations, with queries extracted from MIDI data. Doraisamy and Ruger [13] studied n-grams with both pitch and rhythm information in the context of QBH, using simulated errors. Bainbridge, Dewsnip and Witten [14] also explored n-grams along with state-based matching and dynamic programming. A synthetic error model was used, and their paper also proposed using fast n-gram search as a filter to reduce the size of the search by slower methods. Our work uses audio queries from human subjects, and further explores the possibility of a 2-stage search using n-grams and dynamic programming.

3. OBTAINING THE BEST PERFORMANCE WITH NOTE-INTERVAL SEARCH

We want to explain the wide variation in observed performance of note-based dynamic programming searches. In order to understand what factors are critical, we tune a search system in different ways and compare the results. We begin by designing a note-based dynamic programming algorithm called "NOTE-SIMPLE". NOTE-SIMPLE is quite similar to the note-interval algorithm, but simpler in some ways and can take into account different sets of parameters and representations for evaluations.

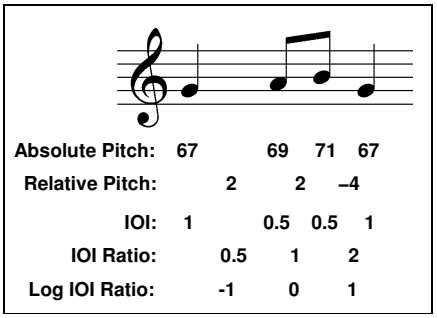

Figure 1. Pitch Interval and IOI Ratio calculation.

The NOTE-SIMPLE algorithm can perform search on different representations of melodic sequence. *Pitch* and *Rhythm* are the two main components defining each note N_i in a melodic sequence $S = N_1 N_2 \cdots N_n$. As shown in Figure 1, the *Pitch* component can be expressed in two ways:

1. The absolute pitch in MIDI key values:
   $P_{abs}(N_i) \in \{1, 2, ...127\}$, where $1 \le i \le n$
2. The relative pitch, or pitch interval:
   $P_{rel}(N_i) = P_{abs}(N_i) - P_{abs}(N_{i-1})$, where $1 < i \le n$
   $P_{rel}(N_1) = 0$

Similarly, there are three different kinds of representation for the *Rhythm* component of N_i:

1. The inter-onset-interval (IOI):
   $T_{IOI}(N_i) = t_{onset}(N_{i+1}) - t_{onset}(N_i)$, where $1 \le i < n$
   $T_{IOI}(N_n) = t_{offset}(N_n) - t_{onset}(N_n)$
2. The IOI Ratio (IOIR), which is the ratio between the IOI values of two succeeding notes:
   $T_{IOIR}(N_i) = \frac{T_{IOI}(N_i)}{T_{IOI}(N_{i-1})}$, where $1 < i \le n$
   $T_{IOIR}(N_1) = 1$
3. The Log IOI Ratio (LogIOIR) [15], the logarithm of the IOI Ratio:
   $T_{LogIOIR}(N_i) = \log(T_{IOIR}(N_i))$, where $1 \le i \le n$

IOIR and LogIOIR are usually quantized by rounding the values to their closest integers [15]. However, we did not quantize in our experiments, and this might be an option to explore in the future.

Like the note-interval or other common dynamic programming search algorithms, NOTE-SIMPLE computes the melodic edit distance D(A, B) between two melodic sequences $A = a_1 a_2 \cdots a_m$ and $B = b_1 b_2 \cdots b_n$ by filling the matrix $(d_{0...m,0...n})$. Each entry $d_{i,j}$ denotes the minimal melodic edit distance between the two prefixes $a_1 \ldots a_i$ and $b_w \ldots b_j$ $(1 \le w \le j)$.

We use a classical calculation pattern for the algorithm as shown:

for $1 \le i \le m$ and $1 \le j \le n$,

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + w(a_i, \phi) & (deletion) \\ d_{i,j-1} + w(\phi, b_j) & (insertion) \\ d_{i-1,j-1} + w(a_i, b_j) & (replacement) \\ \{d_{i-k,j-1} + w(a_{i-k+1}, \ldots, a_i, b_j), 2 \le k \le i\} & (consolidation) \\ \{d_{i-1,j-k} + w(a_i, b_{j-k+1}, \ldots b_j), 2 \le k \le j\} & (fragmentation) \end{cases}$$

Initial conditions are:

$d_{i,0} = d_{i-1,0} + w(a_i, \phi)$, $i \ge 1$ (deletion)
$d_{0,j} = d_{0,j-1} + w(\phi, b_j)$, $j \ge 1$ (insertion)
and $d_{0,0} = 0$.

In order to simplify the algorithm, we define

$w(a_i, \phi) = k_1 C_{del}$ and $w(\phi, b_j) = k_1 C_{ins}$,

where C_{del} and C_{ins} are constant values representing deletion cost and insertion cost respectively. We also define the replacement weight

$$w(a_i, b_j) = |P(a_i) - P(b_j)| + k_1 |T(a_i) - T(b_j)|,$$

where P() can be $P_{abs}()$ or $P_{rel}()$, and T() is either $T_{IOI}()$ or $T_{LogIOIR}()$. If IOIR is used, then

$$w(a_i, b_j) = \left|P(a_i) - P(b_j)\right| + k_1 \max\left(\frac{T_{IOIR}(a_i)}{T_{IOIR}(b_j)}, \frac{T_{IOIR}(b_j)}{T_{IOIR}(a_i)}\right)$$

k_1 is the parameter weighting the relative importance of pitch and time differences. It is quite possible to be tuned for better performance. But in this experiment, we arbitrarily picked k_1=1 if the Rhythm form is IOI or IOIR and k_1=6 if the form is LogIOIR. Those values achieved reasonable results in our initial experiments.

The equations for computing fragmentation and consolidation [16, 17] are only used in the calculation pattern when the *Rhythm* input is in IOI form, as our previous experiments based on IOI [16] proves that fragmentation and consolidation are beneficial to the performance. We do not use fragmentation or consolidation for the *Rhythm* input in IOIR or LogIOIR form, since fragmentation and consolidation do not really make sense when dealing with ratios.

If the algorithm is computed on absolute pitches, the melodic contour will be transposed 12 times from 0 to 11 in case the query is a transposition of the target. [16] Also the contour will be scaled multiple times if IOI is used. Both transposition and time scaling increase the computing time significantly.

Testing was performed using the MUSART testbed [7], which has two sets of queries and targets. Database 1 is a collection of Beatles songs, with 2844 themes, and Database 2 contains popular and traditional songs, with 8926 themes. In this section, we report results using Database 2, which is larger. In most instances, we use the *mean reciprocal rank* (MRR) to evaluate search performance. For example if there are only two queries for which the correct targets are ranked second and fourth, the MRR is (1/2 + 1/4)/2 = 0.375. Thus, MRR ranges from zero (bad) to one (perfect).

Table 1 lists some results obtained from the NOTE-SIMPLE algorithm for different representations of melodic sequence. For each of these tests, the insertion and deletion costs were chosen to obtain the best performance. The combination of Relative Pitch and Log IOI Ratio results in the best performance.

Table 1. Retrieval results using various representations of pitch and rhythm.

Representations	MRR
Absolute Pitch & IOI	0.0194
Absolute Pitch & IOIR	0.0452
Absolute Pitch & LogIOIR	0.0516
Relative Pith & IOI	0.1032
Relative Pitch & IOIR	0.1355
Relative Pitch & LogIOIR	0.2323

The relationship between the insertion and deletion costs is another interesting issue to be investigated. Table 2 shows the results from different combinations of insertion and deletion costs. Note that these values are scaled by k_1 = 6.

The main point of Table 1 and Table 2 is that design choices have a large impact on performance. NOTE-SIMPLE does not perform quite as well as Pardo's note-interval search algorithm [7], perhaps because his note-interval search normalizes the replacement cost function to behave as a probability distribution. Further tuning might result in more improvements. Overall, we conclude that dynamic programming is quite sensitive to parameters. Best results seem to be obtained with relative pitches, Log IOI Ratios, and carefully chosen insertion and deletion costs. Previous work that did *not* use these settings may have drawn false conclusions by obtaining poor results.

Table 2. Retrieval results using different insertion and deletion costs.

C_{ins} : C_{del}	MRR
0.5 : 0.5	0.1290
1.0 : 1.0	0.1484
2.0 : 2.0	0.1613
1.0 : 0.5	0.1161
1.5 : 1.0	0.1355
2.0 : 1.0	0.1290
0.5 : 1.0	0.1742

C_{ins} : C_{del}	MRR
1.0 : 1.5	0.2000
0.2 : 2.0	0.2194
0.4 : 2.0	0.2323
0.6 : 2.0	0.2323
0.8 : 2.0	0.2258
1.0 : 2.0	0.2129

4. REIMPLEMENTATION AND TESTING OF THE CUBYHUM SEARCH ALGORITHM

CubyHum is a QBH system developed at Philips Research Eindhoven. It uses a dynamic programming algorithm for melodic search. We are particularly interested in comparing the performance of its search algorithm with our algorithms. Thus we re-implemented the CubyHum search algorithm (a.k.a. CUBYHUM) in our system, following the published description [8].

CUBYHUM uses relative pitches in semitones; however, it further quantizes the relative pitches into 9 integers: –4, –3, … 3, 4. The calculation pattern of the search algorithm is much more complex than the one used in NOTE-SIMPLE. However, it does not achieve very satisfying results even compared to our simple dynamic programming algorithm NOTE-SIMPLE. (See Table 3.) This seems to be in conflict with the common notion that complex calculation patterns yield better performance, but it is consistent with the notion that performance can vary tremendously with different design choices.

Table 3. Performance of CubyHum compared to NOTE-SIMPLE, as measured by MRR.

Algorithm	Database 1	Database 2
CUBYHUM	0.0229	0.0194
NOTE-SIMPLE	0.1221	0.2323

5. N-GRAMS FOR QUERY-BY-HUMMING

An important problem with all of the approaches we have described so far is their performance in a large database. Even with substantial optimization, our fastest

algorithm, the NOTE-SIMPLE algorithm, would run for many minutes or even hours on a database with a million songs (and perhaps 10 million themes). One possible solution is to use n-grams, which allow an index to be constructed to speed up searching.

5.1. Two-stage search.

It is unnecessary for the n-gram approach to work as well as note-interval matching or other techniques. The important thing is for n-grams to have very high recall with enough precision to rule out most of the database targets from further consideration. A more precise search such as the note-interval matcher can then be used to select a handful of final results. This two-stage search concept is diagrammed in Figure 2.

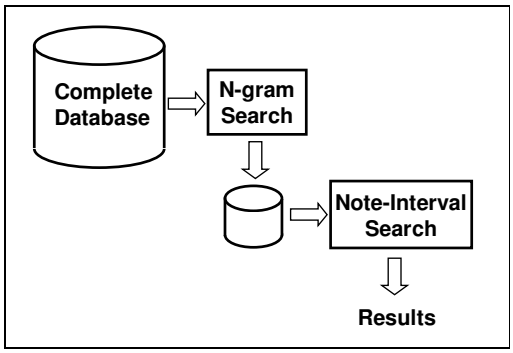

Figure 2. A two-stage search using n-gram search for speed and note-interval search for precision.

5.2. N-gram search algorithms

Our n-gram search operates as follows: The audio query is transcribed into a sequence of notes as in the note-interval search, and note intervals are computed. Pitch intervals are quantized to the following seven ranges, expressed as the number of half steps:

< –7, –7 to –3, –2 to –1, unison, 1 to 2, 3 to 7, >7

IOI Ratios are quantized to five ranges separated by the following four thresholds: $\sqrt{2}/4$, $\sqrt{2}/2$, $\sqrt{2}$, $2\sqrt{2}$. Thus, the nominal IOI Ratios are ¼, ½, 1, 2, and 4. These fairly coarse quantization levels, especially for pitch, illustrate an important difference between n-grams and other approaches. Whereas other searches consider of small differences between query and target intervals, n-grams either match exactly or not at all. Hence, coarse quantization is used so that small singing or transcription errors are not so likely to cause a mismatch.

N-grams are formed from sequences of intervals. Figure 1 illustrates how a trigram is formed from a sequence of four notes. Note that the IOI for the last note is not defined, so we use the last note's duration instead. In Figure 1, the trigram formed from pitch intervals and IOI Ratios is <2, 0.5, 2, 1, –4, 2>.

A set of n-grams is computed for the query and for each target by looking at the *n* pitch intervals and IOI Ratios beginning at each successive note. For example, trigrams would be formed from query notes 1, 2, and 3, notes 2, 3, and 4, notes 3, 4, and 5, etc.

To compute similarity, we basically count the number of n-grams in the query that match n-grams in the target. Several variations, based on concepts from text retrieval [18, 19] were tested. The following are independent design decisions and can be used in any combination:

1. Count the number of n-grams in the query that have a match in the target (e.g. if an n-gram occurs 3 times in the query and twice in the target, the score is incremented by 2). Alternatively, weight each n-gram in the query by the number of occurrences in the target divided by the number in the query (e.g. if an n-gram occurs 3 times in the query and twice in the target, the score is incremented by 2/3). This is a variation of term frequency (TF) weighting.
2. Optionally weight each match by the inverse frequency of the n-gram in the whole database This is known as Inverse Document Frequency (IDF) weighting, and we use the formula log(N/d), where N is the total number of targets, and d is the number of targets in which the n-gram occurs.
3. Optionally use a locality constraint: consider only target n-grams that fall within a temporal window the size of the query.
4. Choose n-gram features: (a) Incorporate Relative Pitch and IOI Ratios in the n-grams, (b) use only Relative Pitch, or (c) use only IOI Ratios.
5. Of course, n is a parameter. We tried 1, 2, 3, and 4.

5.3. N-gram performance on vocal queries

Although we were unable to explore all 96 permutations of these design choices, each choice was tested independently in at least several configurations. The best performance was obtained with *n = 3,* using combined IOI Ratios and Relative Pitch (3 of each) in n-grams, not using the locality constraint, using inverse document frequency (IDF) weighting, and not using term frequency (TF) weighting. This result held for both MUSART databases.

Figure 3 show results for different n-gram features and different choices of *n.* As can be seen, note interval trigrams (combining pitch and rhythm information) work the best with these queries and targets.

5.4. N-grams in a two-stage search

The n-gram search (MRRs of 0.09 and 0.11 for the two databases) is not nearly as good as the note-interval search algorithm (MRRs of 0.13 and 0.28), but our real interest is the potential effectiveness of a two-stage system.

To study the possibilities, consider only the queries where a full search with the note-interval search algorithm will return the correct target ranked in the top 10. (If the second stage is going to fail, there is little reason to worry about the first stage performance.) Among these "successful" queries, the average rank in the n-gram search tells us the average number of results an n-gram search will need to return to contain the correct

target. Since the slower second-stage search must look at each of these results, the possible speed-up in search time is given by:

$$s = N / \bar{r},$$

where s is the speedup ($s \geq 1$), N is the database size, and $\bar{r}$ is the mean (expected value of) rank. Table 4 shows results from our two databases. Thus, in Database 2, we could conceivably achieve a speedup of 3.45 using n-grams to eliminate most of the database from consideration.

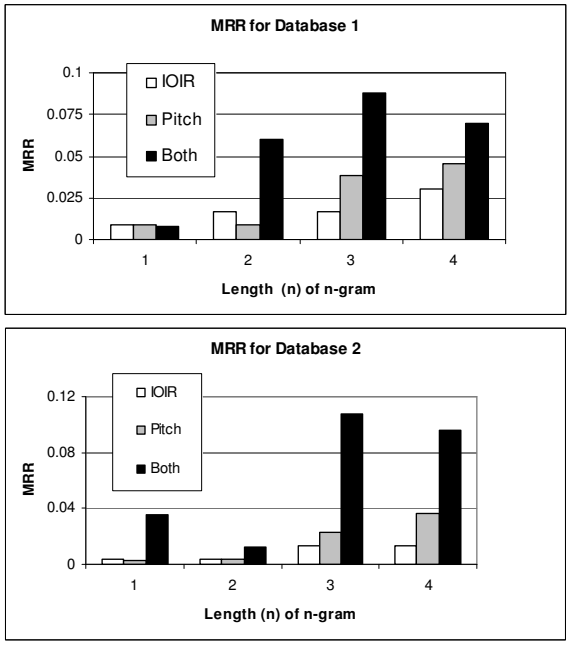

Figure 3. N-gram search results on Databases 1 and 2.

Table 4. Fraction of database and potential speedup.

Database	$\bar{r}/N$	s
1 (Beatles)	0.49	2.06
2 (General)	0.29	3.45

Of course, we have no way to know in advance where the n-gram search will rank the correct target, and n-gram searching takes time too, so this theoretical speedup is an upper bound. Another way to look at the data is to consider how results are affected by returning a fixed fraction of the database from the n-gram search. Again, considering only queries where the second stage search ranks the correct target in the top 10, we can plot the number of correct targets returned by the first-stage n-gram search as a function of the fraction of the database returned.

As seen in Figure 4, n-gram search is significantly better than random, but somewhat disappointing as a mechanism to obtain large improvements in search speed. If the n-gram search returns 10% of the database, which would reduce the second-stage search time tenfold, about 50% to 65% of the correct results will be lost. Even if the n-gram search returns 50% of the entire database, the number of correct results is still cut by 25% to 40%. These numbers might improve if the n-gram search returns a variable number of results based on confidence.

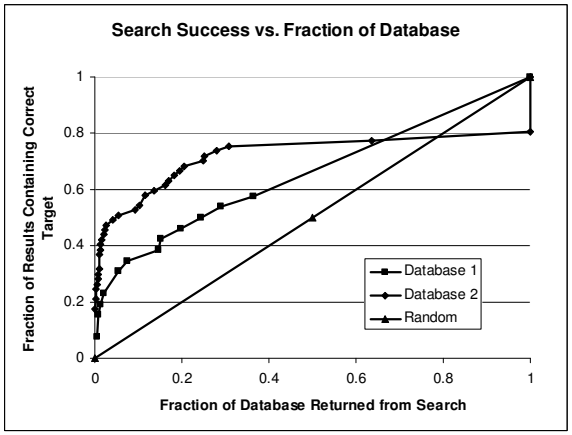

Figure 4. Performance of the best n-gram search showing the proportion of correct targets returned as a function of the total number of results returned.

The n-gram search fails on a substantial number of queries that can be handled quite well by slower searches. Bainbridge, et al. say "It is known that music-based n-gram systems are computationally very efficient and have high recall…" [14], but with our data, we see that about 40% of the correct Database 1 targets are ranked last, and about 20% of the correct Database 2 targets are ranked last. A last-place ranking usually means that the target tied with many other targets with a score of zero (no n-grams matched). In the event of a tie, we report the highest (worst) rank. Overall, our results with "real" audio queries suggest that singing and transcription errors place significant limits on n-gram system recall.

6. SUMMARY AND CONCLUSIONS

Query-by-Humming systems remain quite sensitive to errors in queries, and in our experience, real audio queries from human subjects are likely to be full of errors and difficult to transcribe. This presents a very challenging problem for melodic similarity algorithms.

By studying many configurations of note-based dynamic programming algorithms, we have determined that (1) these algorithms are quite competitive with the best techniques including melodic contour search and HMM-based searching, but (2) parameters and configuration are quite important. Previous work has both *overestimated* the performance of note-based DP searching by using simple tasks and *underestimated* the performance by failing to use the best configuration.

We re-implemented the CubyHum search algorithm and found that it performs poorly compared to a simpler but well-tuned note-based DP algorithm.

We also studied the use of n-grams for query-by-humming. Overall, n-grams perform significantly worse than other melodic-similarity-based search schemes. The main difference is that n-grams (in our implementation) require an exact match, so the search is not enhanced by

the presence of approximate matches. Also, intervals must be quantized to obtain discrete n-grams, further degrading the information.

We considered the use of n-grams as a "front end" in a two-stage search in which a fast indexing algorithm based on n-grams narrows the search, and a high precision algorithm based on dynamic programming or HMMs performs the final selection. We conclude that there is a significant trade-off between speed and precision. We believe our results form a good indication of what is possible with n-gram searches applied to "real world" queries of popular music from non-musicians.

7. FUTURE WORK

Indexing for melodic contours remains an interesting and open question. N-grams suffer because at least small errors are very common in sung queries. A better understanding of melody recognition by humans might suggest new strategies. In particular, a better sense of what we recognize and remember might allow vast reductions in database size, perhaps even eliminating the need for indexing. Query-by-Humming with "real" audio queries is difficult even for the best of known algorithms. While the average user may be unable to produce effective queries for unconstrained music searches on the web, there is probably room for creative applications of QBH in personal media management systems, music assistants, education, and entertainment systems yet to be envisioned.

8. ACKNOWLEDGEMENTS

The authors would like to acknowledge the MUSART team for building the testbed used in this study. Special thanks are due to Bill Birmingham, George Tzanetakis, Bryan Pardo, and Collin Meek. Stephen Downie provided helpful comments and references that guided our work with n-grams. This paper is based on work supported by the National Science Foundation under grant IIS-0085945.

9. REFERENCES

[1] R. B. Dannenberg, W. P. Birmingham, G. Tzanetakis, C. Meek, N. Hu, and B. Pardo, "The MUSART Testbed for Query-by-Humming Evaluation," *Computer Music Journal*, vol. 28, pp. 34-48, 2004.

[2] W. Hewlett and E. Selfridge-Field, "Melodic Similarity: Concepts, Procedures, and Applications," in *Computing in Musicology*, vol. 11. Cambridge: MIT Press, 1998.

[3] R. B. Dannenberg, "An On-Line Algorithm for Real-Time Accompaniment," in *Proceedings of the 1984 International Computer Music Conference*, Paris, San Francisco:International Computer Music Association, 1985, pp. 193-198.

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

[5] C. Meek and W. P. Birmingham, "Johnny Can't Sing: A Comprehensive Error Model for Sung Music Queries," in *ISMIR 2002 Conference Proceedings*, Paris, France, Paris:IRCAM 2002, pp. 124-132.

[6] B. Pardo, W. P. Birmingham, and J. Shifrin, "Name that Tune: A Pilot Studying in Finding a Melody from a Sung Query," *Journal of the American Society for Information Science and Technology*, vol. 55, 2004.

[7] R. B. Dannenberg, W. P. Birmingham, G. Tzanetakis, C. Meek, N. Hu, and B. Pardo, "The MUSART Testbed for Query-by-Humming Evaluation," in *ISMIR 2003 Proceedings of the Fourth International Conference on Music Information Retrieval*, Baltimore, MD, Baltimore:Johns Hopkins University 2003, pp. 41-50.

[8] S. Pauws., "CubyHum: A Fully Operational Query-by-Humming System," in *ISMIR 2002 Conference Proceedings*, Paris, France, Paris:IRCAM 2002, pp. 187-196.

[9] M. Lesaffre, K. Tanghe, G. Martens, D. Moelants, M. Leman, B. D. Baets, H. D. Meyer, and J.-P. Martens, "The MAMI Query-By-Voice Experiment: Collecting and Annotating Vocal Queries for Music Information Retrieval," in *ISMIR 2003 Proceedings of the Fourth International Conference on Music Information Retrieval*, Baltimore, MD, Baltimore:Johns Hopkins University 2003, pp. 65-71.

[10] J. S. Downie and M. Nelson, "Evaluation of a simple and effective music information retrieval method," in *Proceedings of ACM SIGIR* 2000, pp. 73-80.

[11] J. S. Downie, *Evaluating a Simple Approach to Music Information Retrieval*, Ph.D. Thesis.: Faculty of Information and Media Studies, University of Western Ontario, 1999.

[12] A. Uitdenbogerd and J. Zobel, "Melodic Matching Techniques for Large Music Databases," in *Proceedings of the 7th ACM International Multimedia Conference*:ACM 1999, pp. 57-66.

[13] S. Doraisamy and S. Ruger, "A Comparative and Fault-tolerance Study of the Use of N-grams with Polyphonic Music," in *ISMIR 2002*, Paris, France 2002, pp. 101-106.

[14] D. Bainbridge, M. Dewsnip, and I. H. Witten, "Searching Digital Music Libraries," in *Digital Libraries: People, Knowledge, and Technology: 5th International Conference on Asian Digital Libraries*, Singapore, New York:Springer-Verlag 2002, pp. 129-140.

[15] B. Pardo and W. P. Birmingham, "Encoding Timing Information for Musical Query Matching," in *ISMIR 2002 Conference Proceedings*, Paris, France, Paris:IRCAM, October 13-17 2002, pp. 267-268.

[16] N. Hu and R. B. Dannenberg, "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 2002, pp. 301-307.

[17] M. Mongeau and D. Sankoff, "Comparison of Musical Sequences," in *Melodic Similarity Concepts, Procedures, and Applications*, vol. 11, *Computing in Musicology*, W. Hewlett and E. Selfridge-Field, Eds. Cambridge: MIT Press, 1990.

[18] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*: Addison-Wesley, 1988.

[19] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*: McGraw-Hill, 1983.