5-2006

A Comparative Evaluation of Search Techniques for Query-by-Humming Using the MUSART Testbed

Roger B. Dannenberg
Carnegie Mellon University, rbd@cs.cmu.edu

William P. Birmingham
Grove City College

Bryan Pardo
Northwestern University

Ning Hu
Google, Inc.

Colin Meek
Microsoft Corporation

See next page for additional authors

Published In
Journal of the American Society for Information Science and Technology, 58, 5, 687- 701.

A Comparative Evaluation of Search Techniques for Query-by-Humming Using the MUSART Testbed

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

Abstract

Query-by-Humming systems offer content-based searching for melodies and require no special musical training or knowledge. Many such systems have been built, but there has not been much useful evaluation and comparison in the literature due to the lack of shared databases and queries. The MUSART project testbed allows various search algorithms to be compared using a shared framework that automatically runs experiments and summarizes results. Using this testbed, we compared algorithms based on string alignment, melodic contour matching, a hidden Markov model, n-grams, and CubyHum. Retrieval performance is very sensitive to distance functions and the representation of pitch and rhythm, which raises questions about some previously published conclusions. Some algorithms are particularly sensitive to the quality of queries. Our queries, which are taken from human subjects in a fairly realistic setting, are quite difficult, especially for n-gram models. Finally, simulations on query-by-humming performance as a function of database size indicate that retrieval performance falls only slowly as the database size increases.

Introduction

In "Query-by-Humming" systems, the user sings or hums a melody and the system searches a musical database for matches. Query-by-Humming can be thought of as an automated version of the game "Name That Tune." In addition to providing song titles, Query-by-Humming systems offer an interesting interface possibility for portable MP3 players, for digital music search through the web, and for kiosks offering to sell music. Query-by-Humming is an alternative to text searches for title, composer, and artist in digital music libraries. A particularly interesting feature of Query-by-Humming is that the user is not required to understand music notation or any music-theoretical description of the sought-after content.

Aside from practical benefits, Query-by-Humming offers many intrinsically interesting challenges for researchers. At the heart of any Query-by-Humming system is some model of melodic similarity. There is never an exact match between a sung query and the desired content, so a Query-by-Humming system must ignore the superficial details of the query waveform and work at more abstract levels in order to make meaningful comparisons. Many issues arise relating to the production and perception of music, including how users remember melody, the limitations of amateurs in the vocal production of melody, our perception of melody, and the nature of melodic similarity. There are also many issues relating to algorithms and databases, including probabilistic models of error and melodic distance, efficient search algorithms, and system architecture. Finally, there are interesting intellectual property and business issues that bear on what musical databases can contain, what information they can provide, and what services can be offered.

We have studied many of the technical aspects of Query-by-Humming in the MUSART project. In our investigations, we constructed a number of different search systems, assembled several experimental databases, and collected many different queries. We also built various tools for estimating musical pitches in sung queries, for transcribing queries into sequences of notes, and for automatically extracting musical themes from standard MIDI files. After working on these various experimental systems for some time, we found that our work was becoming fragmented, with incompatible software versions that could not be compared under controlled conditions. This state of affairs mirrors what we observe in the research community at large: While there are many different research systems with published performance

Published as: Roger B. Dannenberg, William P. Birmingham, Bryan Pardo, Ning Hu, Colin Meek, George Tzanetakis. "A Comparative Evaluation of Search Techniques for Query-by-Humming Using the MUSART Testbed." *Journal of the American Society for Information Science and Technology* **58** (3) (February 1, 2007).

measurements, these measurements cannot be compared. In order to learn more about search algorithms and their performance, we need carefully developed tests.

To solve this problem, at least within our project, we built the MUSART Testbed, a framework for testing and comparing various approaches to Query-by-Humming. We adapted our previous work (standalone experimental software to study various aspects of the problem) to operate within the testbed, enabling us to make fair comparisons between different approaches. The MUSART Testbed includes a database of songs, collections of recorded audio queries, programs to extract data from queries and song files, and a collection of search algorithms. Tests can be run automatically to evaluate the performance of different algorithms with different collections of songs and queries.

We have compared a number of approaches to Query-by-Humming and obtained some surprising results. Some sophisticated and computationally expensive search techniques do not perform much better than one that is simpler and much faster. We found search performance is very sensitive to the choice of distance functions (the likelihood that a “state” will be transcribed, given a “state” in the melody). The conclusions of many previous studies must be considered carefully in this new light.

Due to the problems of collecting and annotating large databases, our test database is limited in size. To examine issues of scalability, we estimate performance as a function of the database size. In all cases we have examined, the database precision falls roughly according to $1/\log(x)$ where x is the database size. This is encouraging because this function is flat, meaning performance falls very slowly with increases in database size.

Related Work

Query-by-Humming can be considered a special case of melodic search given a query that approximates at least a portion of the melody of a target song. For example, the query may be a text-encoded sequence of pitches, rhythms, or a note sequence recorded from a digital piano keyboard. Melodic search has been the focus of many studies, and concepts of melodic similarity are presented in (Hewlett & Selfridge-Field, 1998). References on melodic search can be found at www.music-ir.org. Various approaches have been taken to the problem of identifying similar melodic sequences. String-matching approaches using dynamic programming (Sankoff & Kruskal, 1983) have been popular (Bainbridge, Dewsnip, & Witten, 2002; Bainbridge, Nevill-Manning, Witten, Smith, & McNab, 1999; McNab, Smith, Witten, Henderson, & Cunningham, 1996; Pauws., 2002) and work well with melody, as we shall see. Another approach uses n-grams, which are widely used in text retrieval and allow efficient indexing. (Clausen, Englebrecht, & al., 2000; Doraisamy & Ruger, 2002, 2003; Downie & Nelson, 2000; Tseng, 1999; Uitdenbogerd & Zobel, 1999) Another set of techniques rely on statistical models including Markov and hidden Markov models. (Durey & Clements, 2001; Hoos, Rentz, & Gorg, 2001; Jin & Jagadish, 2002; C. Meek & W.P. Birmingham, 2002; Pardo, Birmingham, & Shifrin, 2004). It should be noted that there is, at best, a weak distinction between melodic search based on hidden Markov models and search based on string matching. (Durbin, Eddy, Krogh, & Mitchison, 1998; Hu & Dannenberg, 2002; Bryan Pardo & William P. Birmingham, 2002)

Hsu, *et al.* (Hsu & Chen, 2001; Hsu, Chen, Chen, & Liu, 2002) describe their Ultima project, created for the study, evaluation and comparison various music search algorithms. Three search algorithms for melody matching are described and compared using automatically generated queries. Bainbridge, Dewsnip, and Witten (2002) describe their workbench for symbolic music information retrieval in the Greenstone digital-library architecture. In the reported experiments, a large folksong database was searched using synthetically generated queries where subsequences of songs in the database were altered to simulate human errors. Like the MUSART testbed, both the Ultima project and Greenstone workbench implement several search techniques, provide a database of melodies, and support experimentation and comparison of different techniques. Synthetic queries allow researchers to control query lengths and error counts, but leave open the question of performance with real human queries.

Query Processing and Music Representation

Audio signals cannot be compared directly, as even two “identical” melodies from the same instrument or vocalist will have little if any direct correlation between their waveforms. Therefore, melodic search must be performed on a higher-level or more abstract representation. In typical audio recordings, the mixture of vocals, harmony, bass, and drums results in a complex waveform that cannot be separated automatically into symbolic (e.g., notes) or even audio components. In spite of progress in extracting melody from recorded audio (Goto, 2000), current systems are not sufficiently robust to enable effective music searching. Therefore, Query-by-Humming systems assume a symbolic representation of music in the database. We use a database of MIDI files, which describe the pitch[1], starting time, duration, and relative loudness of every note in a piece of music. Most MIDI files are organized into a set of tracks, where each track contains the notes for one instrument. These representations are normally prepared by hand. They can also be extracted automatically from machine-readable music notation if available. In our testbed, we use MIDI files found on the Web.

Queries, on the other hand, contain only the sound of one voice, which makes analysis much more tractable. Depending on the search algorithm, we must obtain a sequence of fundamental frequency estimates or a sequence of notes from the query. Our system analyzes the fundamental frequency of the signal every 10 milliseconds, using an enhanced autocorrelation algorithm (Tolonen & Karjalainen, 2000) with a 50 millisecond window size. In this step, we report "no pitch" when the amplitude is below threshold or when there is no clear fundamental indicated by the autocorrelation.

To transcribe the query into notes, we must separate the frame sequence into notes. Notes begin when there is a sequence of five or more frames with frequency estimates that fall within the range of one semitone. In other words, a note must exhibit a steady pitch. The note ends when the pitch changes or when no pitch is detected. Pitch is also quantized from a continuous scale to the 12-pitches-per-octave Western scale. We assume that singers do not have an absolute pitch reference, but that singers attempt to sing exact equal-temperament pitches. To avoid an absolute pitch reference, we quantize *pitch intervals* to the nearest integer number of semitones. In practice, many singers make significant pitch errors, and the pitch estimation can introduce further errors, so it is important for the search algorithms to take this into account.

[1] Nomenclature varies somewhat across disciplines. Scientifically, "pitch" denotes a percept closely related to the physical property of "fundamental frequency," the rate of vibration. Musically, "pitch" often refers to the musical scale, e.g. C4, C#4, D4, etc., where the number refers to the octave, and C4 is "middle C." We will use pitch in the musical sense because it is more concise than "the chromatic scale step corresponding to the fundamental frequency."

Extracting Melody from MIDI Files

In addition to preprocessing the audio queries, we also preprocess the database of MIDI files. Complete MIDI files include much more than just melodies. In a typical song, the melody is repeated several times. There is also harmony, drumming, a bass line, and many notes may be performed by more than one instrument, e.g., if a violin and flute may play the same notes, the MIDI file will have separate copies of the notes for the violin and flute. Searching entire MIDI files will take more time and could result in spurious matches to harmonies, bass lines, or even drum patterns . For both speed and precision, we extract melodies from MIDI files and search the melodies. (Tseng, 1999)

We developed a program called *ThemeExtractor* (Meek & Birmingham, 2001) to find musical themes in MIDI files. The principle is simple: melodies are almost always repeated several times, so if we find significant sequences that repeat, we will locate at least most of the melodies. To enhance the accuracy of melody identification, repeating melodic patterns are analyzed to obtain various features that include register (average pitch height), rhythmic consistency, and other features that help identify patterns that are "interesting." (It is common to find repeating patterns of just one or two pitches, used for rhythm or accompaniment, and these should not qualify as "melody.") The features are weighted to form a score for each melodic pattern. The patterns with the highest scores are returned as the most significant melodic phrases or themes of the piece. Tests indicate that ThemeExtractor finds a very high percentage of themes labeled by hand. (Meek & Birmingham, 2001)

Searching for Melody

The MUSART testbed maintains a collection of themes that have been extracted from the database using the ThemeExtractor program. We call these themes the *targets* of the search. To process a query, the testbed first processes the audio as described earlier to obtain a symbolic representation of the query. Then, the query is compared to every target in the database. Each comparison results in a *melodic similarity* value, and the targets are sorted according to this similarity. In some searches, *distance* is reported rather than *similarity*, but this only requires that we sort in the opposite direction.

To measure system performance, we need to know the rank of the correct target(s) in the sorted list resulting from each search. The testbed includes a list of correct matches for each query. Correct matches are determined manually by listening and using file names. Call *r* the rank of the correct target for a query.

$$P_{rel}(N_i) = P_{abs}(N_i) - P_{abs}(N_{i-1}),\ \text{where } 1 < i \le n \qquad P_{rel}(N_1) = 0 \qquad \text{Equation 3}$$

Here, *r* = 1 indicates the correct target was ranked first. The mean right rank for a trial is the average value for *r* over all queries in the trial. This measure can be sensitive to poorly ranking outliers. We can capture the same information in a manner that is less sensitive to outliers by using the *mean reciprocal rank* (MRR). The reciprocal rank is 1/*r*, and the mean reciprocal rank is just the mean of the reciprocal rank over all queries in the trial. If the system always ranks a correct answer highest (rank 1), the MRR will be 1.

$$T_{IOI}(N_i) = t_{onset}(N_{i+1}) - t_{onset}(N_i),\ \text{where } 1 \le i < n \qquad T_{IOI}(N_n) = t_{offset}(N_n) - t_{onset}(N_n) \qquad \text{Equation 4}$$

If the system gives random similarity values to targets, the MRR will be roughly log(*N*)/*N*, and the worst possible MRR is 1/*N*, where *N* is the number of targets in the database. These relations are shown in Equation 1.

$$1 \ge MRR = \frac{\sum_{q=1}^{Q} \frac{1}{r_q}}{Q} \ge \frac{1}{N} \qquad \text{Equation 1}$$

We have implemented five different search algorithms for comparison in the MUSART testbed.

$$P_{abs}(N_i) \in \{1,2,...127\},\ \text{where } 1 \le i \le n \qquad \text{Equation 2}$$

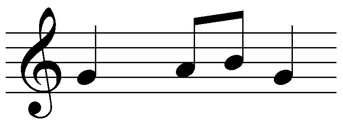
These algorithms are now described.

Note Interval Matching

The *Note Interval* matcher treats melodies as strings and uses dynamic-programming techniques to align two strings, resulting in a similarity score.

Figure 1 illustrates the representations used by this matcher. The *Pitch* component can be expressed in two ways:

1. The absolute pitch in MIDI key values (it cannot be 0 as 0 means silence).
2. The relative pitch, which is the pitch interval between two adjacent notes that are expressed in absolute pitches.



Absolute Pitch:	**67**		**69**		**71**		**67**
Relative Pitch:		**2**		**2**		**−4**	
IOI:	**1**		**0.5**		**0.5**		**1**
IOI Ratio:		**0.5**		**1**		**2**	
Log IOI Ratio:		**-1**		**0**		**1**	

Figure 1. Pitch Interval and IOI Ratio calculation.

The advantage of relative pitch is that transposition (singing the query in a different key) amounts to an additive offset to absolute pitch, so there is no effect on relative pitch. We say that relative pitch is *transposition invariant.*

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

1. The inter-onset-interval (IOI), which is the time difference between two adjacent note onsets:
   Rather than IOI, one might consider *duration,* that is, toffset(Ni) – tonset(Ni). In our experience, IOI is a better representation for search than note duration. The beginning of a note (tonset) is perceptually more salient than the ending (toffset), and therefore the IOI seems to be a better indication of the perceived musical rhythm. Furthermore, the note offset times detected from the query are not accurate. Using IOI rather than note duration amounts to extending each note as necessary to remove any silence before the next note. (For the last note, we use duration for lack of anything better.)
2. The IOI Ratio (IOIR), which is the ratio between the IOI values of two succeeding notes:

$$T_{IOIR}(N_i) = \frac{T_{IOI}(N_i)}{T_{IOI}(N_{i-1})}, \text{ where}$$

$$1 < i \le n \qquad \text{Equation 5}$$

$$T_{IOIR}(N_1) = 1$$

3. The Log IOI Ratio (LogIOIR) (Bryan Pardo & W.P. Birmingham, 2002), the logarithm of the IOI Ratio:

$$T_{LogIOIR}(N_i) = \log(T_{IOIR}(N_i)),$$

$$\text{where } 1 \le i \le n \qquad \text{Equation 6}$$

Both IOIR and LogIOIR have the nice property that they are invariant with respect to tempo. Thus, even if the query tempo is faster or slower than the target, the IOIR and LogIOIR values will still match.

A *Note Interval* combines a pitch interval with a Log IOI Ratio to form a <*Pitch, Rhythm*> pair. In the Note Interval matcher, LogIOIR is quantized to the nearest integer, and 5 LogIOIR values ranging from −2 to +2 are used. (Bryan Pardo & W.P. Birmingham, 2002) For example, the full representation of the melody in Figure 1 would be: <<2, −1>, <2, 0>, <−4, 1>>. In the Sensitivity Studies section, we consider the effect of using other representations, but for now, we will consider only the best-performing configuration (Pardo et al., 2004) using pitch intervals and LogIOIRs as the melodic representation.

Using the classic dynamic-programming approach, the Note Interval matcher computes the melodic similarity 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 ($d_{1\ldots m, 1\ldots n}$). Each entry $d_{i,j}$ denotes the maximum melodic similarity 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 ≤ i ≤ m and 1 ≤ j ≤ n,

$$d_{i,j} = \max \begin{cases} 0 & (\textit{local alignment}) \\ d_{i-1,j-1} + \textit{substitutionReward}(a_i, b_j) & \\ & (\textit{replacement}) \\ d_{i-1,j} - \textit{skipCost}_a & (\textit{deletion}) \\ d_{i,j-1} - \textit{skipCost}_b & (\textit{insertion}) \end{cases}$$

Equation 7

In this formulation, *substitutionReward*(*a*, *b*) is the reward (similarity) between note intervals *a* and *b*. The *skipCost*$_a$ and *skipCost*$_b$ are penalties for either deleting a note interval *a* from the target or inserting the note interval *b* found in the query. *Local alignment* means any portion of the query is allowed to match any portion of the target. This is implemented efficiently simply by replacing any negative value by zero in the matrix, effectively ignoring any prefix of the query or target that does not make a good match. The overall similarity is taken to be the maximum value in the matrix, which may ignore contributions from query and target suffixes. The calculation of $d_{i,j}$ is illustrated in Figure 2, showing how the computation can be organized as a matrix computed from left-to-right and top-to-bottom.

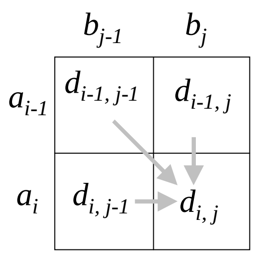


Figure 2. Calculation pattern for Simple Note Interval matcher.

N-Gram Matching

A standard approach in text retrieval is the use of n-grams, tuples of N words. An index from n-grams to documents can be constructed, allowing the search algorithm to consider only those documents that contain n-grams of interest. In music, n-gram-based searches can use pitch intervals and/or inter-onset intervals to retrieve songs.

N-gram approaches generally assume that the query contains specific n-grams that have a high probability of occurrence in the correct target and a low probability of occurrence elsewhere. When multiple n-grams can be obtained from the query, the probability of finding most or all of these in any but the correct target can become vanishingly small. This is particularly true of text searches, where words are discrete symbols, and of symbolic music searches, where pitches and intervals are discrete. With sung queries, the singing is far from perfect, and transcriptions to symbolic data contain substantial errors. There is a tension between choosing long n-grams (large N) to decrease the probability of matching incorrect targets, and choosing short n-grams (small N) to decrease the probability that singing and transcription errors will prevent any matches to the correct target.

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 allow the consideration of small differences between query intervals 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.

There are two important differences between note interval matching and n-gram search. First, there is the obvious algorithmic difference. Secondly, there is the fact that n-grams are based on exact matching whereas note interval matching considers approximate matches through the use of skip costs and replacement costs. We have not tried to study these two factors independently, as both degrade search performance and both are necessary to achieve the efficiency gains of n-gram search.

N-grams are formed from sequences of intervals. 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. 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 (2, 2, and −4) and IOI Ratios (0.5, 1, and 2) before quantization is

<2, 0.5, 2, 1, −4, 2>.

To compute similarity, we count the number of n-grams in the query that match n-grams in the target. Several variations, based on concepts from text retrieval (Salton, 1988; Salton & McGill, 1983) 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 (once matched, n-grams are not reused; if there are q copies of *ngram* in the query and t copies in the target, the score is incremented by $\min(q, t)$). Alternatively, weight each n-gram in the query by the number of occurrences in the target (i.e. increment the score by t). 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.

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. Even if thousands of results are returned, a more precise search such as the note-interval matcher can be used to select a handful of final results. This two-stage search concept is diagrammed in Figure 3. We implemented a flexible n-gram search to explore this possibility.

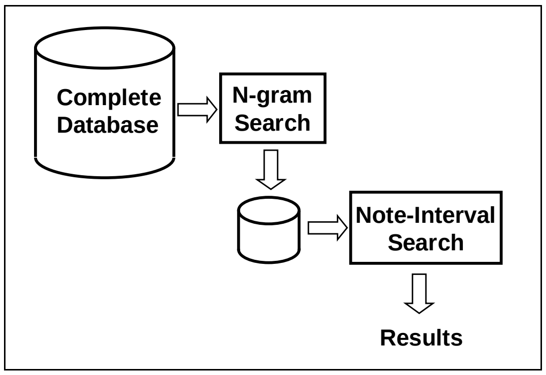


Figure 3. A two-stage search using n-gram search for speed and note-interval search for precision.

Melodic Contour Matching

Our *Melodic Contour* matcher (Mazzoni & Dannenberg, 2001) uses dynamic time-warping to align the pitch contour of the query with that of the target. This is similar in spirit to the Note Interval matcher operating with absolute pitch, but there are significant differences. Most importantly, the Melodic Contour matcher uses fixed-sized *frames* of about 100 ms duration rather than notes, which can have any duration. The advantage of this *contour* representation is that there is no need to segment queries into discrete notes. Segmentation is a difficult problem when users do not clearly articulate notes, and each segmentation error effectively inserts or deletes a spurious note into the query. Users also have problems holding a steady pitch and making a quick pitch transition from note to note, resulting in possible pitch errors in the transcribed queries. In contrast, the contour representation treats pitch as an almost continuous time-varying signal and ignores note onsets, so it is more tolerant of transcription errors.

To match queries to targets, the target melodies are also split into equal-duration frames. If a frame overlaps two or more notes, the frame is assigned to the pitch of the note that occupies the greatest amount of time in the frame. Notes are extended to the onset of the next note, eliminating rests.

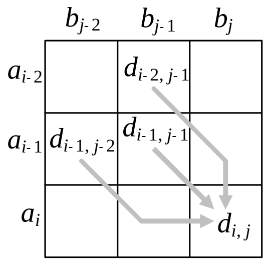
To deal with tempo variation, we time-scale the target data by factors of 0.5, 1.0, and 2.0 (finer-grain scaling does not seem to help.) Then, we use dynamic time warping (DTW) to align the query frames to the target frames. We have experimented with different variations of DTW. (Hu and Dannenberg, 2002) In some variations, extreme tempo changes are allowed, allowing a fairly good match by selectively skipping large portions of target melodies to select pitches that match the query. This, of course, can lead to false matches. We had better results with the calculation pattern shown in Figure 4, which limits the local tempo change to a factor of two speed-up or slow-down.

Another important aspect of this matcher is that we want to allow a query to match any portion of the target, without requiring a full match. The DTW algorithm used allows the skipping of any prefix and any suffix of the target with negligible differences in run time. This is accomplished by setting $d_{0,j}=0$ for all j, allowing the match to begin anywhere in the target without penalty, and taking the best value in the final column, allowing the match to end anywhere without penalty. Transposition is handled by calculating the contour similarity with 24 different transpositions in 1/24th octave steps. All pitches are mapped to a single octave by using the actual pitch number modulo 12 in case the melody has been transposed more than one octave.

HMM Matching

One of the limitations of the matchers described so far is that they do not have a detailed model of different types of errors. In contrast, musicians will recognize common problems in queries: perhaps the singer's pitch will get progressively flatter, a single high note will be sharp, or the tempo might slow down or speed up. While the previously described matchers model a few error types and tolerate others, the *Hidden Markov Model* matcher can model many more error types. This allows a less-than-perfect but still plausible query to achieve a high similarity score against its target. In general, there are two types of errors modeled: "local" errors are momentary deviations in pitch or tempo, and "cumulative" errors are the result of a trend.

"Johnny Can't Sing" (JCS) models local and cumulative error in pitch and rhythm. A detailed description of the training and matching algorithms used by JCS are published (Meek & Birmingham, 2004). JCS is an extended hidden Markov model (HMM) (Rabiner, 1989) in which the target and query notes are associated through a series of *hidden states.* Each hidden state is defined by a triple $s_i = \langle E[i], K[i], S'[i] \rangle$, where $E[i]$ is the "edit" type that accounts for errors such as skipping, inserting, merging, or splitting notes, $K[i]$ is the "key" that accounts for transpositions in pitch between the target



$$d_{i,j} = \mathbf{min} \left\{ \begin{array}{c} d_{i-1,j-1} \\ d_{i-2,j-1} + w(a_{i-1}, b_j) \\ d_{i-1,j-2} + w(a_i, b_{j-1}) \end{array} \right\} + w(a_i, b_j),$$

(1 = i = m, 1 = j = n)

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

and the query, and *S'*[*i*] is the "speed" that accounts for differences in tempo and duration.

Observations are defined by the duple $o_t = \langle Pitch, Rhythm \rangle = \langle P[t], R[t] \rangle$, which is the pitch and rhythm (IOI) observed in the query. As in the standard hidden Markov model, the observations occur with some probability that depends only on a corresponding hidden state, and the hidden state depends only on the previous hidden state. Once the HMM is defined, the goal of the matching algorithm is to determine the probability of a match between the target and the query. This involves searching over all possible sequences of hidden states (which specify sequences of note alignment, transposition, and tempo) and then determining the probability of each observation of pitch and rhythm.

Given the many possible edit types, transposition amounts, and tempo mappings, the number of states is huge. JCS deals with this problem by assuming that the three components of the hidden state are somewhat independent. Figure 5 illustrates a conventional HMM structure on the left and the JCS distributed state structure on the right. In the distributed state structure, the edit type (*E*) depends only on the previous edit type. The "key" or transposition (*K*) depends on the previous key and the current edit type. A pitch observation (*P*) depends on the current edit type and key but is independent of tempo. The probability of the observation depends on the discrepancy between the observation and the expectation established by the transposition and the target's pitch. Similarly, the observed rhythm (*R*) depends on the edit type (*E*) and tempo (*S'*), but not on transposition. By factoring the model into components and assuming independence, the size of the model is greatly reduced, making similarity ratings much more tractable.

Figure 6 shows an example of how the components of a state are combined. The edit (*E*) indicates that the first two target notes are joined to match the next note of the query. The key (*K*) indicates a transposition of +2 semitones, and the tempo (*S'*) indicates a scaling of 1.25. The resulting transformed event represents the most likely observation given the current hidden state(s) <*E, K, S'*>. The actual observation (shown in black) is higher in pitch and shorter in IOI. It is said to have a pitch error of +1 (semitones) and a rhythm error of 0.8 (a factor).

The HMM were trained in a variety of ways, which are described in another paper. (Meek and Birmingham, 2004) We summarize the training by mentioning that we tested various training/test set partitioning strategies: random, by singer (different singers in the test and training sets), and by song.

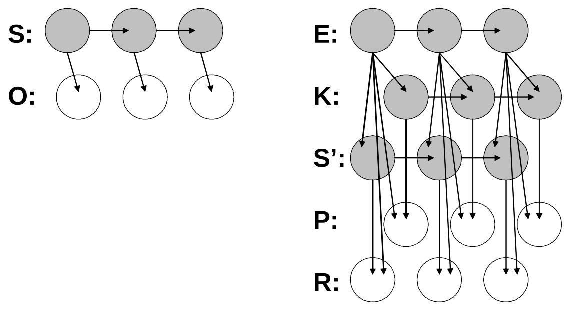


Figure 5: Dependency schemata for basic and distributed state HMM representations. Shaded circles indicate "hidden" states, and white circles indicate fully observable states. Arrows indicate probabilistic dependencies in the models, which evolve over time from left to right. The conventional HMM structure is shown at the left, and an alternative distributed state structure is shown at the right. Note that the new state S' is dependent on the states E and K. In addition, the observations are dependent on the state S'. Thus, rather than one state corresponding to one observations, the definition of "state" and "observation" are now expanded to include several variables.

The CubyHum Matcher

CubyHum is a QBH system developed at Philips Research Eindhoven. It uses a edit distance for melodic search. An interesting aspect of the system is

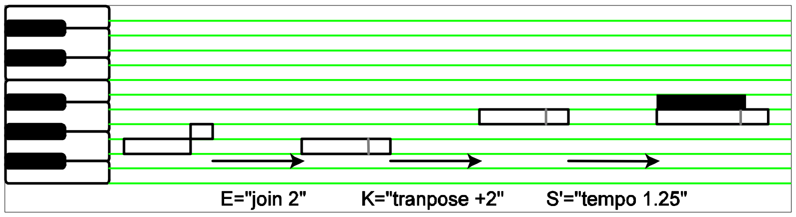


Figure 6: Interpretation of a JCS state.

that it models errors such as modulation or tempo change using elaborate rules in the dynamic-programming calculation. Since dynamic programming is a special case of HMMs, CubyHum can be viewed as an alternative to the HMM model of JCS described above. It would be interesting to discover that the elaborate mechanisms of JCS could be replaced by a much faster algorithm. Therefore, we are particularly interested in comparing the performance of CubyHum with our algorithms. We re-implemented the CubyHum search algorithm in our system, following the published description (Pauws, 2002).

Like the Note Interval matcher, CubyHum uses a representation based on pitch intervals and IOI ratios. Pitch intervals are quantized to nine ranges and assigned values of -4 to 4. IOI ratios are not quantized. The CubyHum dynamic programming calculation pattern computes an *edit distance* that includes five rules to handle various error types:

- The *modulation or no error* rule adds a penalty for pitch interval differences and IOI ratio differences between corresponding intervals in the query and the target. Since it is based on intervals, this error models a transposition or a tempo change, or both. An isolated pitch or duration error would be modeled as two modulations or two tempo changes, respectively.
- The *note-deletion* rule can be applied when two intervals of the query sum to one corresponding interval of the target, implying there is an extra note in the query. This is essentially a consolidation rule: the two notes of the query are consolidated as if the pitch of the first were changed to that of the second, resulting in a longer note with the correct pitch interval. There is a fixed penalty for the extra note, and an additional penalty for the difference in IOI ratios.
- The *note-insertion* rule is similar to *note deletion*; it handles the case where two pitch intervals in the target sum to match a single interval in the query. The two target notes are consolidated into one note, and a fixed penalty plus a penalty based on IOI ratios is added.
- The *interval-deletion* rule skips an interval in the target, and
- The *interval-insertion* rule inserts an interval in the target, adding a fixed penalty plus a penalty based on IOI ratios.

The CubyHum description (Pauws, 2002) also presents methods for query transcription and an indexing method for faster search. In order to focus on melodic similarity algorithms and to make results comparable, we use the same query transcription for testing all of our search algorithms, and we did not implement the CubyHum indexing method.

Results of the Comparison

Testing was performed using the MUSART testbed (Dannenberg et al., 2003), 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 most instances, we use the mean reciprocal rank (MRR) to evaluate search performance.

There are two sets of queries, corresponding to the two databases. Query Set 1 was collected by asking 10 subjects to sing the “most memorable” part after presenting one of 10 Beatles songs. No instructions were given as to how to sing, so some subjects sang lyrics. Subjects were allowed to sing more than one query if they felt the first attempt was not good, so there are a total of 131 queries. Most of these can be recognized, but many do not correspond very well to the intended targets. Subjects skipped sections of melody, introduced pitch errors, and sang with pitch inflections that make pitch identification and note segmentation difficult. There are also noises from touching the microphone, self-conscious laughter, and other sounds in the queries. The total size is about 75 MB for audio, but only 28 KB for transcriptions. The Database 1 MIDI files occupy about 5.4 MB and the themes are 0.96 MB.

Query Set 2 is the result of a class project in which students were asked to find and record volunteers who sang songs from memory (the target song was not played for the subject before recording the query). A total of 165 usable queries were collected, and all correspond to songs in Database 2. These queries suffer from the same sorts of problems found in Query Set 1. The total size is about 125 MB for audio, and 45 KB for transcriptions. The Database 2 MIDI files occupy about 22 MB, and the themes total 2.2 MB. Table 1 has additional statistics on the queries and databases.

Performance of Melodic Comparison Algorithms

Table 2 shows the results of running Query Set 1 against the 2844 themes of Database 1. One can see that the matchers are significantly different in terms of search quality. At least with these queries, it seems

Table 1: Statistics on Queries and Databases.

	Query Set 1	Query Set 2	Database 1	Database 2
Number of Queries and Songs	155	131	868	258
Total Notes	3658	2527	365065	112771
Number of Themes	-	-	8902	2844
Mean No. Themes	-	-	10	11
Std. Dev. No. Themes	-	-	5.4	3.6
Mean No. Notes per Query or Theme	24	19	41	40
Std. Dev. No. Notes per Query or Theme	14	7	34	31
Mean Duration (Queries and Themes)	10 s	9.3 s	20 s	19 s
Std. Dev. Duration (Queries and Themes)	6.5 s	3.2 s	18 s	16 s

Table 2 Mean. Reciprocal Rank (MRR) for Query Set 1.

Search Algorithm	MRR
Note Interval	0.134
N-gram	0.090
Melodic Contour	0.210
Hidden Markov Model	0.270
CubyHum	0.023

Table 3. Mean Reciprocal Rank (MRR) for Query Set 2.

Search Algorithm	MRR
Note Interval	0.282
N-gram	0.110
Melodic Contour	0.329
Hidden Markov Model	0.310
CubyHum	0.093

that better melodic similarity and error models give better search performance.

Table 3 shows the results of running Query Set 2 against the 8926 themes of Database 2. All five algorithms performed better on this data than with Query Set 1, even though there are many more themes. Unlike Table 2, where the algorithms seem to be significantly different, the top three algorithms in this test have similar performance, with an MRR near 0.3. 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.

Both CubyHum and n-gram search performed considerably less well than the others. Pauws says of CubyHum (Pauws, 2002), "In our singing experiment, we found that the percentage of errors allowed is in the range of 20-40%." It is likely that the MUSART queries are often much worse than this, but recall that all of these queries were sung by subjects who were presumably trying to produce a reasonable query. The n-gram search is discussed at greater length in the next section.

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 search performance (Mazzoni & Dannenberg, 2001). For example, in one test, 65% of correct targets were ranked 1 using melodic contour, while only 25% were ranked 1 using a note-based search. 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). Because the Note Interval approach seems to be important, we explore the design space of a family of related matchers in later sections.

Performance on Synthetic Data

One might also ask how well these algorithms perform using synthetic data. For each query, we constructed a synthetic query by extracting the first *n* notes of the first theme of a matching target, where *n* is the number of detected notes in the query. Thus, the synthetic queries are perfect partial matches. Using dynamic programming on pitch intervals only, the MRR for Query Set 2 (the larger of the two) is 0.85. With n-grams of length 3 using pitch intervals and IOI ratios, the MRR for Query Set 2 is 0.87. These MRRs are much higher than with any of the search algorithms using real query data. The MRRs are still less than perfect (a perfect MRR is 1.0) for several reasons. First, the quality of some queries is so low that only a few notes can be extracted, so even the synthetic queries can be very short. Second, while most recognizable themes are included in the database, we did specifically select these distinctive themes for synthetic queries; instead, many synthetic queries are constructed from very repetitive accompaniment lines that might occur in other targets. Finally, the search algorithms are not optimized to distinguish exact matches, since these do not occur with real data, and often ties for first place led to a rank less than one. Nevertheless, it is clear that the quality of the query is very important. Any query-by-humming system design should consider how to get the best possible queries from its users, regardless of the search algorithm.

Run-Time Performance

Dynamic programming, used in the Note Interval matcher, is an O(*nm*) algorithm, where *n* and *m* are the lengths of the two input strings. Actual run time is of course dependent on hardware and software implementation details, but our software averages about 2.3ms to compare a query to a theme. With about 10 themes per song, run-time is 23ms per song in the database using a 1.8GHz Pentium 4 processor, or 205s for the 8926 songs in Database 2. Nearly all of the run time, measured as “wall time,” is actually CPU time, and we estimate that this could be tuned to run at least several times faster. Additional speed can be obtained using multiple processors.

Our N-Gram implementation does not use a fast indexing scheme, but even with linear search, our implementation averages about 0.2ms per theme, 2.0ms per song, and about 34s to search Database 2. With an index kept in main memory, we would expect the run time to be greatly reduced.

Using N-Grams

Recall that the design space for n-grams is fairly large (96 permutations), so we were unable to test every one. However, each design choice was tested independently in at least several configurations. The best performance was obtained with an n-gram size of n = 3, using combined IOI Ratios and Relative Pitch (three 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.

Figures 7 and 8 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. In general, results are slightly worse without IDF weighting. Results are also slightly worse with TF weighting and with the locality constraint.

N-grams in a two-stage search

The n-gram search results are not nearly as good as those of the note-interval search algorithm (note interval search has MRRs of 0.13 and 0.28 for the two databases, vs. n-grams with MRRs of 0.09 and 0.11), but our real interest is the potential effectiveness of a two-stage system in which n-grams are used to reduce the size of the database to something manageable with a slower but more precise search. There is a tradeoff here: to make searching more efficient, we want to reduce the size of the set returned by the n-gram search, but to insure that the correct result is in that set, we want to increase the size.

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 Equation 8.

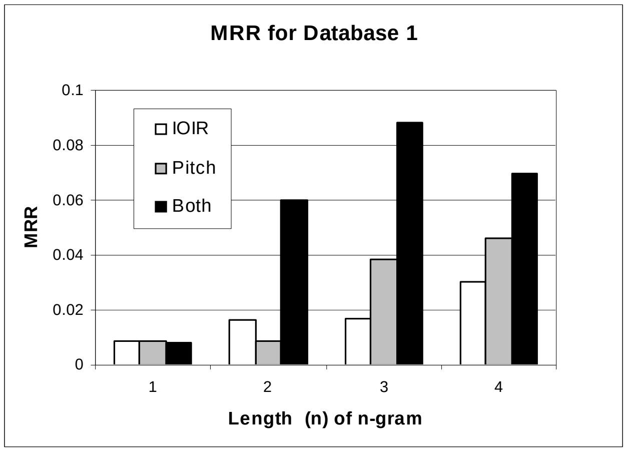


Figure 7. N-gram search results on Database 1.

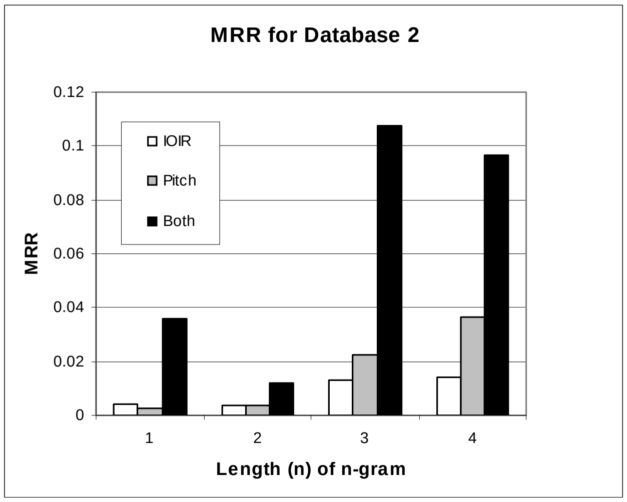


Figure 8. N-gram search results on Database 2.

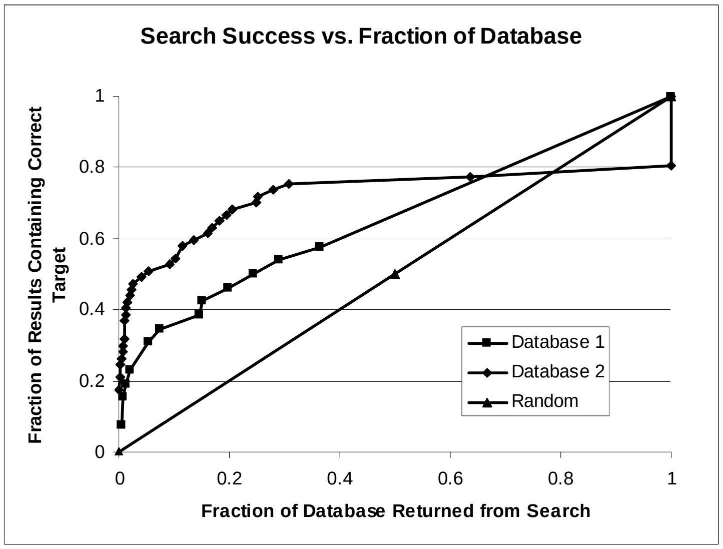


Figure 9. Performance of the best n-gram search showing the proportion of correct targets returned as a function of the total number of results returned.

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

$$s = N / \bar{r} \qquad \text{Equation 8}$$

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.

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 the fraction of the database increases from zero to one, the proportion of correct targets returned goes from zero to one. A search that just picks results at random would form a diagonal on the graph (see Figure 9), whereas a "good" search will have a steep initial slope, meaning that correct targets usually have a low rank.

As seen in Figure 9, n-gram search is significantly better than random, but somewhat disappointing as a mechanism to obtain large improvements in search speed. As can be seen, if the n-gram search returns 10% of the database, which would reduce the second-stage search time ten-fold, 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.

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…" (Bainbridge et al., 2002), 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. This explains why one of the curves in Figure 9 drops below the "random" diagonal. Overall, our results with "real" audio queries suggest that singing and transcription errors place significant limits on n-gram system recall.

Sensitivity Studies

In our studies, we have implemented a number of QBH systems using string-matching algorithms to align melodies and rate their similarity, and many such systems are described in the literature. Our results have been inconsistent, such that seemingly small design changes might result in large changes in performance. We conducted some carefully controlled experiments to examine the impact of various design variations on the performance of string-matching approaches to QBH. This work was conducted within the MUSART testbed framework so that the results could be compared directly to our other QBH search algorithms.

To study design alternatives, we created yet another matcher that we will call the *General Note Interval* matcher. (In a previous publication (Dannenberg & Hu, 2004), we referred to this matcher as "NOTE-SIMPLE.") This matcher is highly configurable, and while "note interval" implies the use of relative pitch and IOI ratios, we can actually specify absolute pitch or any of the rhythm representations (IOI, IOIR, or LogIOIR). We use a classical calculation pattern for the algorithm as shown in Figure 10.

The calculation of $d_{i,j}$ is given by Equation 9. $w(a_i, \phi)$ is the weight associated with the deletion of a_i, $w(\phi, b_j)$ with the insertion of b_j and $w(a_i, b_j)$ with the replacement of a_i by b_j, $w(a_i, b_{j-k+1}, \ldots, bj)$ and $w(a_{i-k+1}, \ldots, a_i, b_j)$ with the fragmentation and the consolidation (Hu & Dannenberg, 2002; Mongeau & Sankoff, 1990) respectively. Fragmentation considers the possibility that the query is not properly segmented; therefore, a single note of the target should be split into multiple notes in order to match a sequence of notes in the query. On the other hand, consolidation combines multiple notes in the target to match a single note in the query. Initial conditions are given by Equation 10.

$$d_{i,0} = d_{i-1,0} + w(a_i, \phi),\ i \geq 1 \quad \text{(deletion)}$$

$$d_{0,j} = d_{0,j-1} + w(\phi, b_j),\ j \geq 1 \quad \text{(insertion)} \qquad \text{Equation 10}$$

$$\text{and } d_{0,0} = 0$$

In order to simplify the algorithm, we define for 1 ≤ i ≤ m and 1 ≤ j ≤ n,

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

Equation 9

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

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

Equation 11

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

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

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

Equation 12

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

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

Equation 13

k_1 is the parameter weighting the relative importance of pitch and time differences. It is quite possible it can 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 (Hu & Dannenberg, 2002; Mongeau & Sankoff, 1990) are only used in the calculation pattern when the *Rhythm* input is in IOI form, as our previous experiments based on IOI (Hu & Dannenberg, 2002) prove 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.

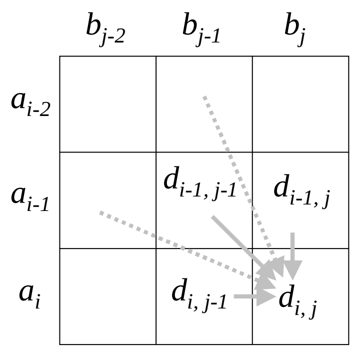


Figure 10. Calculation pattern for General Note Interval matcher.

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. (Hu & Dannenberg, 2002) Also the contour will be scaled multiple times if IOI is used. Both transposition and time scaling will increase the computing time in proportion to the number of transpositions and scale factors used.

The General Note Interval matcher does not completely emulate the Note Interval matcher (Pardo et al., 2004) described earlier. In particular, the Note Interval matcher uses a different calculation pattern that supports *local alignment*, whereas the General Note Interval matcher always performs a forced alignment of the entire query to any contiguous subsequence of the target. Furthermore, the Note Interval matcher normalizes its penalty and reward functions to behave as probability functions.

Table 5 contains some results obtained from the General Note Interval matcher for different representations of melodic sequence using Query Set 2. 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. One surprise is that

Absolute Pitch is consistently worse than Relative Pitch, even though Absolute Pitch searches are performed with the query transposed into all 12 possible pitch transpositions. This must mean that there is often significant modulation (change of transposition) in the middle of queries. With Relative Pitch, such a modulation will only affect one pitch interval and, therefore, contribute less to the overall estimate of melodic distance.

Table 5. Retrieval results using various representations of pitch and rhythm, Query Set 2.

Representations	MRR
Absolute Pitch & IOI	0.019
Absolute Pitch & IOIR	0.045
Absolute Pitch & LogIOIR	0.052
Relative Pitch & IOI	0.103
Relative Pitch & IOIR	0.136
Relative Pitch & LogIOIR	0.232

The relationship between the insertion and deletion costs is another interesting issue to be investigated. Table 6 shows the results from different combinations of insertion and deletion costs using the best representations for pitch and rhythm (Relative Pitch and LogIOIR). Note that these values are scaled by $k_1 = 6$.

Table 6. Retrieval results using different insertion and deletion costs.

$C_{ins} : C_{del}$	MRR
0.5 : 0.5	0.129
1.0 : 1.0	0.148
2.0 : 2.0	0.161
1.0 : 0.5	0.116
1.5 : 1.0	0.136
2.0 : 1.0	0.129
0.5 : 1.0	0.174
1.0 : 1.5	0.200
0.2 : 2.0	0.219
0.4 : 2.0	0.232
0.6 : 2.0	0.232
0.8 : 2.0	0.226
1.0 : 2.0	0.213

The main point of Tables 5 and 6 is that design choices have a large impact on retrieval performance. The General Note Interval matcher does not perform quite as well as the Note Interval matcher algorithm described earlier, perhaps because it normalizes the replacement cost function to behave as a probability distribution and uses local alignment. Further tuning might result in more improvements.

In particular, notice that a change in representation from absolute pitch to relative pitch results in a huge performance increase (from 0.052 to 0.232). Even a change from IOIR to LogIOIR produces a significant performance increase (from 0.135 to 0.232). Insertion and deletion costs are also critical. $C_{ins} : C_{del}$ values of 1.0 : 1.0 result in an MRR of only 0.148 while values of 0.6 : 2.0 yield an MRR of 0.232.

Overall, we conclude that the system 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.

Sources of Error

We have studied where errors arise in the queries. The major problem is that many melodies as sung 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 11). 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. The presence of non-matching melodic material in the query should favor the local alignment policy used in the Note Interval matcher. 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

We have focused our work on melodic similarity. Because some algorithms are quite slow and do not offer a fast indexing method, we have worked with databases that are limited in size.

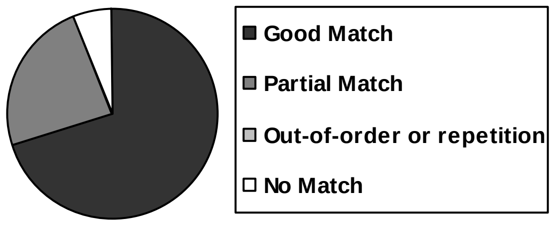


Figure 11: Distribution of query problems. We judged only about half the queries to have a direct correspondence to the correct target.

An interesting question is how does search performance change as a function of the database size. Will search scale up to larger databases? We can explore this question by simulating databases using random subsets of our full database. We could implement this idea by altering the MUSART databases and running a full test on each new database, but this could take a long time. Instead, we compute the similarity rating for every query-target pair just once, and then simulate results with different database sizes.

Assume there are Q queries, T targets, and we have a table of melodic similarity ratings, $S(q, t)$, $0 \le q < Q$ and $0 \le t < T$. We also know the correct target $C(q)$ for each query q. To simulate a search for a query q in a random database of size $N < T$, we first construct the random database consisting of $C(q)$ and N−1 targets randomly selected from $\{0\ldots T-1\}-\{C(q)\}$. Calling this random database R, the rank of the correct target $C(q)$ will be:

$$\text{rank}_{R,N} = 1 + |\{x: x \in R \text{ and } S(q, x) \ge S(q, C(q))\}| \quad \text{Equation 14}$$

In practice, we can "grow" the random database one target at a time. The rank after adding a new target t_{new} will be the same as without the target if t_{new} is less similar to the query than the correct target. Otherwise, the rank will be one greater:

$$\text{rank}_{R,N+1} = \text{rank}_{R,N} + \begin{cases} 0 & \text{if } S(q, t_{new}) < S(q, C(q)), \\ 1 & \text{otherwise} \end{cases}$$

Equation 15

After simulating different database sizes for all queries, we can plot the MRR as a function of database size as shown in Figure 12, which shows an estimated MRR as a function of database size using Note Interval, Melodic Contour, and HMM matchers. Note that all three matchers have similar MRR curves, and all three curves become very flat as the database size grows. This is encouraging because we would like a high MRR even in a much larger database.

The function becomes quite flat as the database grows. Figure 13 shows various analytic functions fitted to the observations. As a baseline, the curve labeled "Random Guess" shows the performance where the search algorithm simply selects a target at random. As expected, the MRR for random guessing approaches zero quickly as the database size grows.

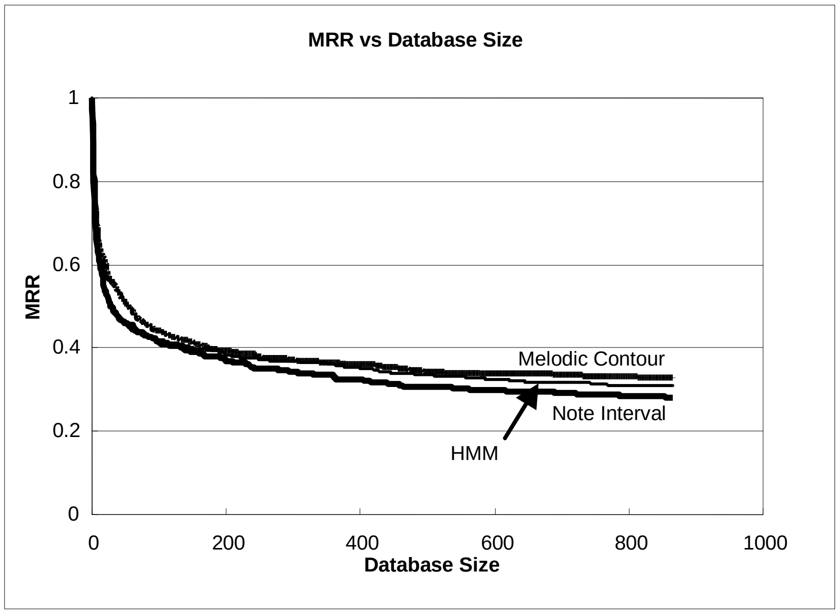


Figure 12: MRR as a function of the number of songs in the database, using subsets of Database 1. Each song represents an average of 10 themes.

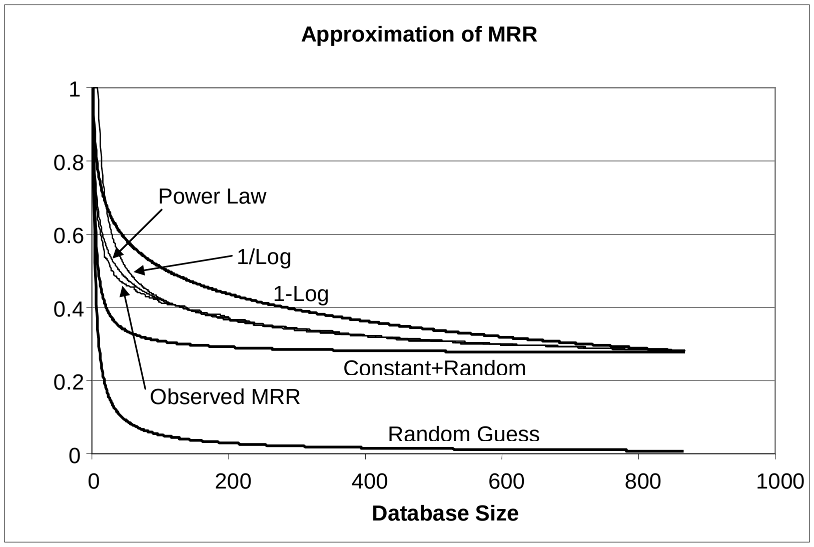


Figure 13: Various models of database scaling with observed data for MRR.

Another model is that certain queries are of good quality and match unambiguously, no matter how large the database, while other queries are so poor that the matcher essentially returns a random guess. This model is labeled "Constant+Random," and it is much flatter than the true MRR curve. Another possibility is 1−*log*(N), labeled "1−Log," but it can be seen that this model does not fit the observed data very well.

Two models that fit the data quite well are the power law model, N^{-p}, labeled "Power Law," and the function $c_1/log(c_2 \cdot N)$, labeled "1/Log." In an earlier study using the fraction of searches that return the correct target at rank 1, we concluded that the "1/Log" function offered the best fit. However, after examining more data and using the MRR rather than Rank = 1 to measure performance, there is no clear difference between these functions. In any case, this exercise in curve-fitting is only a way of describing the observed data, and we cannot claim that either the 1/*log* or the power law function is "correct" in any mathematical sense. Nevertheless, it is encouraging that the performance scales reasonably well.

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 string-alignment 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 string-alignment 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 string-alignment algorithm. Since we did not attempt to replicate the entire CubyHum system including audio transcription, it is possible that there are system-level interactions and dependencies that we have not considered. Also, CubyHum was obviously developed and tuned on better queries and with hand-crafted targets, so it may not be meaningful to test it under a different set of assumptions.

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 edit distance, contour matching, or HMMs performs the final selection. We conclude that there is a significant trade-off between speed and precision.

Of course, we could not explore all possible n-gram approaches, so it is always possible that n-gram search could be improved. (And this was our project's experience with string-alignment-based searching.) Nevertheless, 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.

In conclusion, we have found various algorithms for QBH that perform well with "realistic" audio queries. Nevertheless, the overall performance of QBH systems is quite dependent upon the quality of queries. Users must recall and reproduce melodies at least approximately for any search method to succeed. Another constraint is that the most successful algorithms lack a fast indexing mechanism, so all have a runtime that is linear in the size of the database. The Note Interval search uses a simple and efficient string-alignment approach and therefore seems most promising for applications. We hope that this comparative study using real audio queries will provide some useful insight into the overall problems and potential of query-by-humming search systems.

Acknowledgements

We gratefully acknowledge the support of the National Science Foundation under grant IIS-0085945[2]. The opinions in this paper are solely those of the authors and do not necessarily reflect the opinions of the funding agency. 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 and ISMIR 2004, where some of this information was first presented.

References

Bainbridge, D., Dewsnip, M., & Witten, I. H. (2002). *Searching Digital Music Libraries.* Paper presented at the Digital Libraries: People, Knowledge, and Technology: 5th International Conference on Asian Digital Libraries, Singapore.

Bainbridge, D., Nevill-Manning, C. G., Witten, I. H., Smith, L. A., & McNab, R. J. (1999). *Towards a Digital Library of Popular Music.* Paper presented at the International Conference on Digital Libraries, Berkeley, CA.

Clausen, M., Englebrecht, R., & al., e. (2000). *Proms: A web-based tool for searching in polyphonic music.* Paper presented at the The International Symposium on Music Information Retrieval.

Dannenberg, R. B., Birmingham, W. P., Tzanetakis, G., Meek, C., Hu, N., & Pardo, B. (2003). *The MUSART Testbed for Query-by-Humming Evaluation.* Paper presented at the ISMIR 2003 Proceedings of the Fourth International Conference on Music Information Retrieval, Baltimore, MD.

Dannenberg, R. B., & Hu, N. (2004, October). *Understanding Search Performance in Query-By-Humming Systems.* Paper presented at the ISMIR 2004 Fifth International Conference on Music Information Retrieval Proceedings, Barcelona.

Doraisamy, S., & Ruger, S. (2002). *A Comparative and Fault-tolerance Study of the Use of N-grams with Polyphonic Music.* Paper presented at the ISMIR 2002, Paris, France.

Doraisamy, S., & Ruger, S. (2003). Robust Polyphonic Music Retrieval with N-grams. *Journal of Intelligent Information Systems, 21*(1), 53-70.

Downie, J. S., & Nelson, M. (2000). *Evaluation of a simple and effective music information retrieval method.* Paper presented at the Proceedings of ACM SIGIR.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis*: Cambridge University Press.

Durey, A. S., & Clements, M. A. (2001). *Melody Spotting Using Hidden Markov Models.* Paper presented at the 2nd Annual International Symposium on Music Information Retrieval, Bloomington, IN.

Goto, M. (2000, June 2000). *A Robust Predominant-F0 Estimation Method for Real-time Detection of Melody and Bass Lines in CD Recordings.* Paper presented at the Proceedings of the 2000 IEEE International Conference on Acoustics, Speech and Signal Processing.

Hewlett, W., & Selfridge-Field, E. (Eds.). (1998). *Melodic Similarity: Concepts, Procedures, and Applications* (Vol. 11). Cambridge: MIT Press.

Hoos, H., Rentz, K., & Gorg, M. (2001). *GUIDO/MIR - an Experimental Musical Information Retrieval System based on GUIDO Music Notation.* Paper

[2] The majority of this work was done while the authors Birmingham, Meek and Pardo were at The University of Michigan, Ann Arbor, Electrical Engineering and Computer Science Department.

presented at the International Symposium on Music Information Retrieval, Bloomington, IN.

Hsu, J. L., & Chen, A. L. P. (2001). *Building a Platform for Performance Study of Various Music Information Retrieval Approaches.* Paper presented at the 2nd Annual International Symposium on Music Information Retrieval, Bloomington, IN.

Hsu, J. L., Chen, A. L. P., Chen, H.-C., & Liu, N.-H. (2002). *The Effectiveness Study of Various Music Information Retrieval Approaches.* Paper presented at the Conference on Information and Knowledge Management, McLean, VA.

Hu, N., & Dannenberg, R. B. (2002). *A Comparison of Melodic Database Retrieval Techniques Using Sung Queries.* Paper presented at the Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries.

Jin, H., & Jagadish, H. V. (2002). *Indexing Hidden Markov Models for Music Retrieval.* Paper presented at the ISMIR 2002 Conference Proceedings.

Mazzoni, D., & Dannenberg, R. B. (2001). *Melody Matching Directly From Audio.* Paper presented at the 2nd Annual International Symposium on Music Information Retrieval, Bloomington, Indiana.

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.* Paper presented at the Proceedings of the first ACM international conference on Digital Libraries.

Meek, C., & Birmingham, W. P. (2001). *Thematic Extractor.* Paper presented at the 2nd Annual International Symposium on Music Information Retrieval, Bloomington, Indiana.

Meek, C., & Birmingham, W. P. (2004). A comprehensive trainable error model for sung music queries. *Journal of AI Research (JAIR), 22*, 57-91.

Mongeau, M., & Sankoff, D. (1990). Comparison of Musical Sequences. In W. Hewlett & E. Selfridge-Field (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.* Paper presented at the ISMIR 2002 Conference Proceedings, Paris, France.

Pardo, B., & Birmingham, W. P. (2002). *Improved Score Following for Acoustic Performances.* Paper presented at the Proceedings of the 2002 International Computer Music Conference, Gothenburg, Sweden.

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

Pauws, S. (2002). *CubyHum: A Fully Operational Query-by-Humming System.* Paper presented at the ISMIR 2002 Conference Proceedings, Paris, France.

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE, 77*(2), 257-286.

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

Salton, G., & McGill, M. J. (1983). *Introduction to Modern Information Retrieval*: McGraw-Hill.

Sankoff, D., & Kruskal, J. B. (1983). *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Reading, MA: Addison-Wesley.

Tolonen, T., & Karjalainen, M. (2000). A computationally efficient multi-pitch analysis model. *IEEE Transactions on Speech and Audio Processing, 8*(6), 708-716.

Tseng, Y. H. (1999). *Content-based retrieval for music collections.* Paper presented at the Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information, Berkeley, CA.

Uitdenbogerd, A., & Zobel, J. (1999). *Melodic Matching Techniques for Large Music Databases.* Paper presented at the Proceedings of the 7th ACM International Multimedia Conference.