A Comparison of Melodic Database Retrieval Techniques Using Sung Queries

Ning Hu
Entertainment Technology Center
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268 5351
ninghu@cs.cmu.edu

Roger B. Dannenberg
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268 3827
rbd@cs.cmu.edu

ABSTRACT
Query-by-humming systems search a database of music for good matches to a sung, hummed, or whistled melody. Errors in transcription and variations in pitch and tempo can cause substantial mismatch between queries and targets. Thus, algorithms for measuring melodic similarity in query-by-humming systems should be robust. We compare several variations of search algorithms in an effort to improve search precision. In particular, we describe a new frame-based algorithm that significantly outperforms note-by-note algorithms in tests using sung queries and a database of MIDI-encoded music.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *query formulation, retrieval models, search process.*

General Terms
Algorithms, Measurement, Performance, Experimentation.

Keywords
dynamic programming, melodic comparison, melodic search, Music Information Retrieval (MIR), sung query

1. INTRODUCTION
Music Information Retrieval (MIR) is a relatively new area of investigation. The goal of MIR research is to develop new theory and techniques for processing musical information and searching music databases by content. One interesting branch of MIR is sometimes called "Query by Humming". Compared to other data entry methods, humming is considered the easiest way for ordinary non-musicians to express a music query. A client's input device can be a simple microphone connected to a computer. Unfortunately, one feature of sung queries is a high error rate exacerbated by a difficult transcription problem, so robust matching techniques are essential. We are interested in comparing and evaluating melodic matching techniques that compare sung queries to MIDI data. (In the future, we plan to investigate searching audio data as well.)

In this work, we are not particularly concerned with long execution times even though a practical system operating on a large database must be efficient. By ignoring efficiency issues, we can explore a wider range of algorithms. The best of these will serve as a benchmark to evaluate more practical, efficient approaches. Our work indicates that better search is possible by using new measures of melodic similarity.

Our work is part of a larger project, MUSART [1], which integrates techniques for music analysis, representation, abstraction, and search. In this study, we use a database of themes that is automatically constructed from full MIDI files using the MUSART theme extractor [12], and queries are sung by non-experts without any specific instructions for style or articulation. Thus, we believe these results are indicative of "real-world" data. We are in the process of moving our work to a music library to help with data collection and evaluation.

2. RELATED WORK AND BACKGROUND
Music database systems that accept humming queries are becoming increasingly common and more sophisticated. [2, 5, 11] Typically, these databases transcribe sung queries into a sequence of pitches and rhythms. These are then matched to database entries using various string comparison and N-gram algorithms.

Sung queries are known to be difficult to segment into discrete notes. One reason is that people tend to make all kinds of accidental and intentional variations in pitch and duration when they sing. [6, 11] Tempo can vary from roughly half the speed to double, even within one sung query. Furthermore, current methods for pitch extraction and vowel detection make systematic errors. The result is that there can be a large difference between the user's intended query and the query transcribed from a user's vocalization. This makes the melodic search problem difficult.

3. VARIATIONS ON MELODIC SEARCH
Dynamic programming [17] has been applied to melodic comparison [3, 7] and has become a standard technique in music information retrieval. Dynamic programming is popular for music information retrieval because melodic contours can be represented as character strings, thus melodic comparison and search can benefit from the more mature research area of string matching. As the dynamic programming technique is popular for approximate string matching, it is only natural that it be broadly used in the area of melodic search. However, although melodic search is inspired by string matching techniques, it has many properties and practical problems that do not exist in string matching. Here, we

JCDL'02, JULY 13-17, 2002, PORTLAND, OREGON, USA.

Published as: Ning Hu and Roger B. Dannenberg, "A Comparison of Melodic Database Retrieval Techniques Using Sung Queries," in *Joint Conference on Digital Libraries*, (2002), New York: ACM Press, (2002), pp. 301-307.

would like to discuss some important characteristics of melodic search. We can then pursue variations of the dynamic programming techniques to achieve better performance in melodic search.

Dynamic programming is also used in algorithms associated with Markov Models (e.g. the Viterbi algorithm) and Hidden Markov Models (HMM). In fact, many string comparison algorithms and our melodic similarity algorithms can be viewed as special cases of Markov or Hidden Markov Models. [4] The advantage of the Markov formalisms is that arbitrary "edit distances" in string algorithms can be replaced by estimated probabilities in Markov algorithms. We are currently pursuing this direction in hope of improving our search algorithms.

3.1 Edit Distance

When melodies are viewed as strings, one measure of similarity is the number or cost of editing operations that must be performed to make the strings identical. The minimum cost is called the "edit distance." The most common editing operations for melodic comparisons are inserting a note (insertion), deleting a note (deletion) and replacing a note (replacement). Mongeau and Sankoff [13] define two more advanced editing operations: segment one note into multiple notes (fragmentation) and combine multiple notes to form a single note (consolidation). Those five basic operations set up the foundation of a dynamic programming algorithm applied to melodic comparison.

For two sequences $A = a_1, a_2, \ldots, a_m$ and $B = b_1, b_2, \ldots, b_n$, $d_{i,j}$ represents the dissimilarity between $a_1, a_2, \ldots, a_i$ and $b_1, b_2, \ldots, b_j$. The recurrence equation for $1 \le i \le m$ and $1 \le j \le n$ is

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

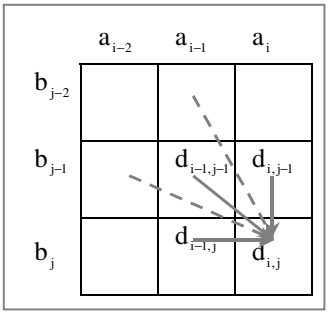


Figure 1. Calculation pattern 1.

The calculation of $d_{i,j}$ is illustrated in Figure 1. $w(a_i, \phi)$ 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 respectively. 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
$d_{0,0} = 0$.

The definitions above represent one possible approach. In fact, there are many variations, some of which will be discussed later.

3.2 Windows and Constraints

Melodic comparison, as just described, compares two strings in their entirety, from beginning to end, and finds the best match even if it requires extensive and unlikely edits. Windows and constraints can be used to rule out some unlikely matches. [9] For hummed queries, it may help to assume that people do not skip many notes, insert many notes, or make drastic tempo changes at any one point. Therefore, "true" matches will match along or near a diagonal of the matrix $d_{i,j}$. We can apply a window to the algorithm, and only the cells inside the window are calculated.

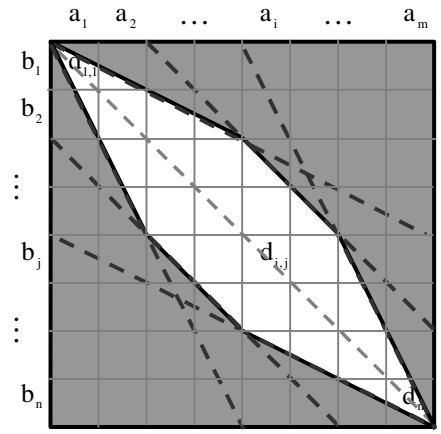


Figure 2. Window within the table.

Figure 2 shows a window applied to the sequence comparison table. The window is basically a diagonal band representing the allowable range of misalignment between the two patterns. The cells in the gray area are assigned a predefined maximum value directly, in order to force the valid paths starting from $d_{1,1}$ to $d_{m,n}$ never to extend outside the window.

While we have described a complete comparison, melodic search is in fact substring matching because queries need not contain every note in a song. In melodic search, the dissimilarity between the query and the compared sequence is actually the smallest dissimilarity between the query sequence and any substring from the compared sequence. Those substrings can start at any position of the sequence and end at any later position, which means they do not have fixed start/end points and the lengths may vary.

Let A be a database sequence and B be a query sequence. To find a match starting at a_1, we use dynamic programming as described above to match all of B, but we do not specify an ending symbol in A. Instead, we take the minimum value (over i) of $d_{i,n}$. A window is used to limit the extent of the search. Assuming the window width grows in proportion to n, the time complexity of this substring comparison will be $O(n^2)$. To find the best match to *any* substring, we can compute the best match starting at each symbol in A, as shown in Figure 3. The time complexity of this algorithm is $O(n^2m)$: each step takes $O(n^2)$ and the window is

moved O(m) times. If a window is not used, this time can be reduced to O(nm) (see below).

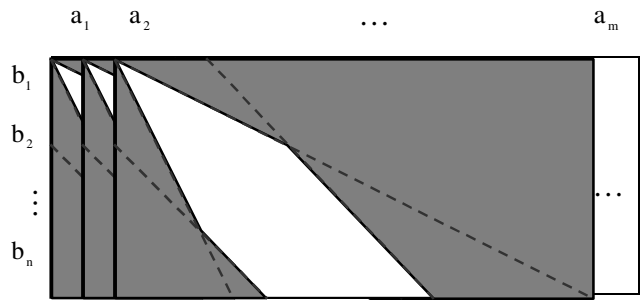


Figure 3. Sliding windows in melodic search.

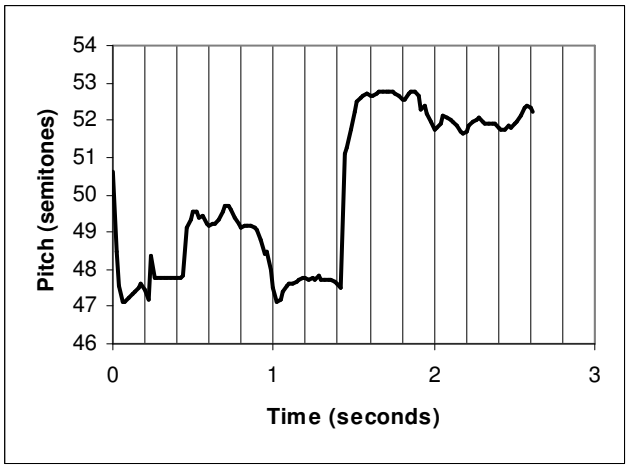
3.3 Frames vs. Notes

We call the note-by-note matching approach *event-based* search to emphasize that notes are discrete events rather than continuous functions of time. As mentioned earlier, there are problems in segmenting (transcribing) an audio query into discrete notes correctly. Furthermore, pitch and duration are two significant attributes of notes, but it is hard to define edit distance other than to consider some linear combination of pitch and duration differences. [16] Because of these problems, the edit distance may not be the best measure of perceptual melodic dissimilarity.

Another representation and search strategy we have explored is called *frame-based* search. Instead of working with discrete notes, the frame-based representation encodes a query rather directly by segmenting the time-varying pitch contour into frames of equal duration. There is no segmentation into notes. [10] This approach is inspired by early speech recognition research [15] and also related to the approach of Nishimura, *et al.* [14]. Figure 4 illustrates a pitch contour, a sequence of pitch estimates used in frame-based algorithms, and a transcribed series of notes used in event-based algorithms.

The frame-based representation has some important advantages over the event-based representation. First, frames do not represent notes explicitly, thus problems relating to note transcription do not arise. These problems include note segmentation (where are the notes?), pitch variation within notes (how do we pick a single pitch for the note?), and note quantization (should pitch be quantized to a musical scale?). Secondly, since the frame sequence includes both the pitch and rhythm information, no weighted combination of pitch and duration distance is required. We also found that incorporating window-like constraints in the frame-based approach is relatively easy. One method we tested can implicitly achieve a window-like constraint within the edit distance equations.

The disadvantage of the frame-based approach is that it is very slow because of much longer query sequences. Also, frames do not carry rhythmic information. If transcription produces good segmentation (for example, if the user can indicate notes by clear articulation or by tapping), then event-based algorithms can take advantage of the additional structure, but frame-based algorithms cannot. In our work, segmentation is generally poor, and this favors the frame-based approach.



Frame representation: 47.1, 47.4, 47.8, 47.8, 47.8, 49.4, 49.2, 49.6, 49.2, 48.4, 47.4, 47.7, 47.7, 47.7, 51.5, 52.6, …

Event representation: (48, 500ms), (49, 500ms), (48, 400ms), (52, 1100ms), …

Figure 4. Original melody contour, frame-based representation and event-based representation.

3.4 Prefix and Suffix

For a substring $A_{sub} = a_i, a_{i+1}, \ldots, a_k$ derived from original sequence $A = a_1, a_2, \ldots, a_m$ $(1 \le i \le k \le m)$, we define the prefix and suffix of that substring to be:

$A_{prefix} = a_1, a_2, \ldots, a_{i-1}$
$A_{suffix} = a_{k+1}, a_{k+2}, \ldots, a_m$

Recall that we want to search for a match between the query sequence and any substring of the compared sequence. Therefore, we want the best match, ignoring some prefix and suffix. In the conventional dynamic programming algorithm, there is a penalty for skipping. As we discussed before, we can avoid the penalty by applying a sliding window that starts at each a_i. Alternatively, we can assign no penalty for skipping the prefix and suffix by making slight changes to the algorithm. [11] The initial conditions of the original melodic comparison algorithm are changed to:

$d_{i,0} = 0, i \ge 0$
$d_{0,j} = d_{0,j-1} + w(\phi, b_j), j \ge 1$

Then, $d_{i,j}$ represents the minimum dissimilarity between the query sequence $b_1, b_2, \ldots, b_j$ and any subsequence of $a_1, a_2, \ldots, a_i$.

Because there are no fixed start points and end points, we cannot restrict computation to a specific window, but the time complexity is O(mn). We call this the "one pass" technique to differentiate with the "sliding window" technique.

3.5 Pitch Transposition

We are not very sensitive to absolute pitch, so the pitches of a melody can be shifted, or *transposed*, by any interval. A melodic sequence can be made invariant with respect to transposition by recording intervals between notes rather than absolute pitches. [18] This representation has the disadvantage of allowing transpositions in the middle of a melody, a major perceptual change, with only a small penalty in terms of edit distance. An alternative is to simply transpose queries into each of 12 possible

keys. To limit the number to 12, we ignore octave transpositions. This still allows shifts of an octave within a melody without penalty, but this is not unreasonable from a perceptual point of view.

For event-based algorithms, we search the database 12 times, once for each possible transposition. Pitches are rounded to the nearest semitone after adjusting sung queries for systematically sharp or flat pitches. The pitches of the frames are not quantized but are represented as floating point values. To limit the cost of search, we ignore octaves here as well and search the database 24 times, transposing the query by quartertones (there are 24 quartertones in one octave).

3.6 Tempo Scaling

In the same way that pitches can be offset without changing the perceptual quality of a melody, time can also be scaled. Thus, it is the relative durations of notes, or perhaps the shape of the melodic contour rather than absolute duration that gives a melody its rhythmic and temporal identity. As with transposition, this problem could be approached with a representation of duration *ratios* rather than absolutes. However, as with the interval representation for pitch, this would allow a drastic change in tempo with only a single insertion or deletion, hence a minor penalty in terms of edit distance. We chose to search the database with numerous time-scaled versions of queries (or alternatively, time-scaled versions of the database entries) to cover a reasonable range of tempos. At least one of these time-scaled versions should be a close match to a correct target in the database. Further minor adjustments are allowed through the dynamic programming algorithm and its associated edit distances.

3.7 Pitch Estimation and Transcription

Pitch estimation is performed using an enhanced autocorrelation algorithm [19] using overlapping windows to estimate 100 fundamental frequencies per second, reporting zero when the amplitude is low or when there is no clear peak fundamental. We found that this works as well as other methods and commercial products that we also tested. [10] For frame-based algorithms, the data is converted to 10 estimates per second by taking the mean of non-zero estimates for that region. For event-based algorithms, transcription is accomplished by first using a histogram method to identify an absolute pitch reference. This minimizes quantization error when fundamental frequency estimates are mapped to discrete pitches. Then, consecutive frames with small pitch differences are merged to form notes. Notes with very low durations are not reported. Source code is available from the authors.

4. EXPERIMENT

To set up our experiment we picked four typical variations of dynamic programming algorithms combining different characteristics discussed above.

4.1 Algorithm 1.

The first one is a simple algorithm that integrates only three most basic operations of dissimilarity comparison: insertion, deletion and replacement. It is based on note sequences, but only considering pitch information, ignoring duration information. So the weight $w(a_i, b_j)$ is defined as the pitch difference between a_i and b_j.

The algorithm is applied with a sliding window, and the calculation for each dissimilarity value $d_{i,j}$ uses the Itakura constraints [8] as shown below:

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

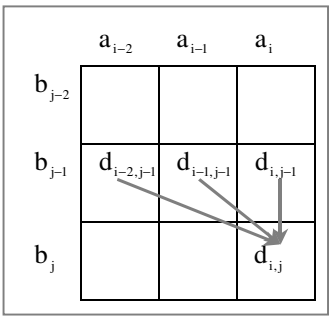


Figure 5. Calculation pattern 2.

The cell relationship for the calculation of $d_{i,j}$ according to the equation shown in Figure 5 represents a typical calculation pattern used in our experiments.

4.2 Algorithm 2.

The second one is also an event-based algorithm but implementing all the five basic operations including fragmentation and consolidation and considering both pitch and duration information, as described by Mongeau and Sankoff [13]. We tried combining "sliding-window" and "one-pass" techniques individually with this algorithm but found little difference in output, except that the one with "one-pass" runs much faster. Probably, the sliding window does not improve precision because the edit distance calculation incorporates duration information and assigns a penalty for tempo variation. In our experiment, this algorithm is applied with the "one-pass" technique. To deal with tempo variation, we scale database durations from 0.5 to 2.0 in steps of 1.08. This was found to significantly enhance the precision of the search by compensating for tempo differences between queries and targets.

4.3 Algorithm 3.

The third algorithm is very similar to the first one, same calculation pattern, same sliding window, same pitch-only weight definition, but based on frame sequences. The query is segmented into 100ms frames, while the compared sequence is segmented according to multiple stretch factors to allow tempo differences.

4.4 Algorithm 4.

The last approach is an improved algorithm invented after studying the previous algorithms and many variations. It is a frame-based algorithm implementing three basic operations—replacement, fragmentation and consolidation, but in an alternative way. The calculation pattern is:

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

$$(1 \le i \le m,\ 1 \le j \le n)$$

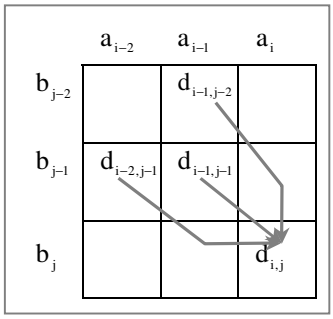


Figure 6. Calculation pattern 3.

As shown in Figure 6, the cell relationship for the calculation is symmetric along the diagonal of $d_{i,j}$. The whole algorithm can be deemed as shrinking the musical contour of either the query or the compared sequence to (locally) double the tempo at certain positions, and then comparing two musical contours exactly. It can also be viewed as assigning a penalty for insertion/deletion.

By observing the calculation pattern 3, we can see that skipping two consecutive frames in either A or B is not permitted except for a prefix and suffix of A. This means that the alignment between A and B will fall within a rhombic pattern shown in Figure 7, shaped much like the window seen earlier. Unlike a window, this constraint is generated implicitly by the edit distance functions, so we get the efficiency of a "one pass" algorithm and constraints similar to a "window" algorithm.

4.5 Database and Queries

For the experiment, we collected and processed 598 MIDI files containing popular songs. These include rock songs, folk songs, and TV theme songs, making it easy to invite non-musicians to sing the theme of a song included in the database. The files contain a total of 1,239,138 notes.

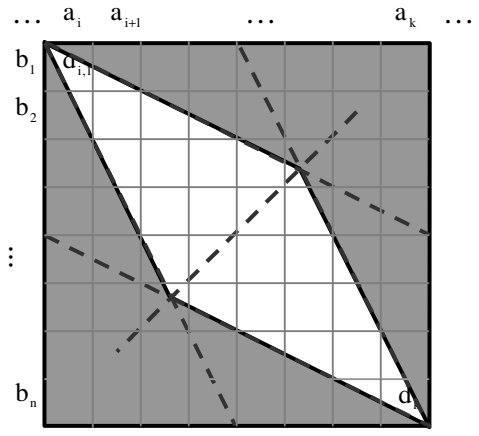


Figure 7. Window-like constraint implicitly set up by calculation pattern 3.

The data is processed using the MUSART thematic extractor [12], which locates the 10 most common phrases or melodies from each original file. After processing, there are 5980 entries in the database with an average length of 22 notes. Although this is still a relatively small number, we believe it is large enough to assess the *relative* quality of different melodic comparison and search algorithms, which is the purpose of this study. Since the original files now have 10 representative melodies, our search algorithms report the best match to any of the 10 themes as the match score for the original file.

Experiments on a smaller, but similar database showed that search performance improved when we searched over themes rather than full files. This is consistent with the fact that the theme extractor does very well at finding the themes that humans identify. And also, subjects are much more likely to hum these themes than some other material contained in the file. By removing non-thematic music from the data, we reduce the chance of matching harmonies, introductions, and material that contains chance similarities to themes of other songs.

We collected 37 queries to assess the system. Subjects were not given any special instructions with respect to how to sing, hum, or whistle a song. This resulted in queries that are generally recognizable but difficult to automatically transcribe accurately. The queries include male and female voices with a range of musical ability, and the queries include humming, singing, and whistling. Table 1 shows the distribution of those queries.

Table 1. Query distribution

	Sing	Whistle	Hum
Female	3	2	0
Male	0	15	17

For each query, we compute a measure of dissimilarity to each entry in the database, and we determine the rank order of the correct database entry for the query. The quality of the algorithm is assessed by counting how many searches return correct songs with a rank order of 1, in the top 10, or in the top 100.

5. RESULTS

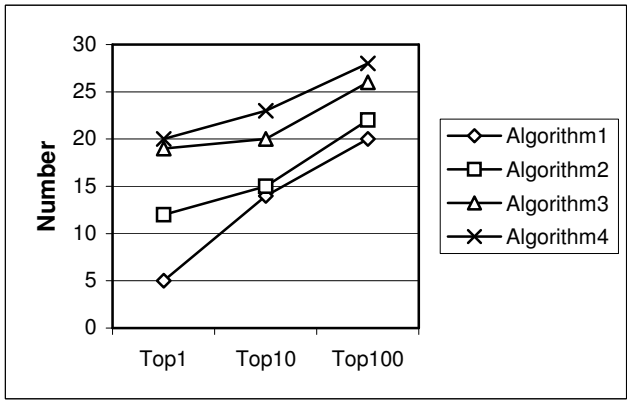


Figure 8. Algorithm comparison chart.

The result shows that frame-based algorithms (Algorithm 3 and 4) perform much better than event-based algorithms (Algorithm 1 and 2). Comparing the two event-based algorithms, the one implementing all five basic operations (Algorithm 2) gets better

results. It is also by far the best among all the event-based algorithms we have tested. We suspect that the consolidation and fragmentation operations make up, in part, for errors in segmentation. Therefore, random segmentation errors do not have such a negative impact on the similarity estimate.

The frame-based algorithm implementing calculation pattern 3 (Algorithm 4) is better than all the other algorithms and it actually runs much faster than the other frame-based algorithm because it only runs the dynamic programming algorithm once across the table. Figure 9 summarizes these properties *qualitatively*. Speed (queries per second) is measured relative to the slowest algorithm; quality is represented using rank order with 1 indicating the best.

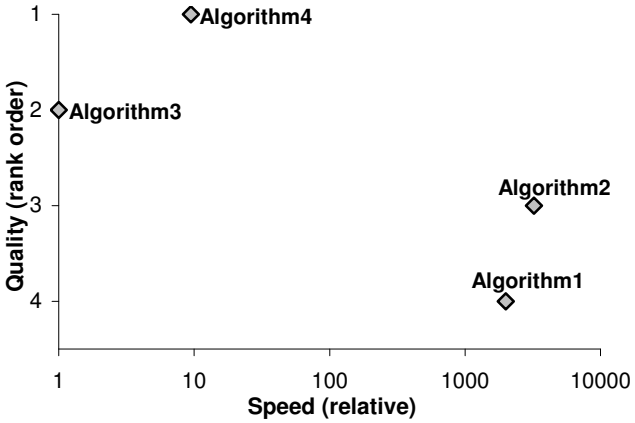


Figure 9. Quality/speed chart for melodic search algorithms.

6. SUMMARY AND CONCLUSION

"Query by Humming" searches melodies in a database for matches to an audio query. There are many sources of error in queries, including the vocalization of the query itself, the estimation of fundamental frequencies in the sound, and the transcription of this frequency contour into a discrete representation.

Algorithms for "Query by Humming" must deal with many issues, including robustness in the face of transcription errors, transposition invariance, overall tempo differences, and local tempo variation. Algorithms must also allow searching for substrings within an overall melody.

To study these issues, we have implemented and evaluated a promising set of search algorithms, including a new class of algorithms we call *frame-based*, because they segment the query into equal-sized time frames. The potential advantages of frame-based algorithms are that they deal with both time and pitch in an elegant manner and they do not rely upon a note segmentation or transcription step that is known to introduce errors. From our experiments, we can conclude that frame-based algorithms outperform the best event-based algorithms in terms of precision. (This conclusion assumes that our audio queries and transcription errors are typical.)

Between the two event-based algorithms, Algorithm 2 performs better. In preliminary work, we tried this algorithm with and without the "consolidation" and "fragmentation" operations, and the performance with these operations is better. Therefore, we conclude "consolidation" and "fragmentation" are important. In contrast, Algorithm 1 uses different constraints and ignores durations.

We found that automatic theme extraction, important for reducing search time, also enhances precision. This is an important finding for making search faster on large databases. Because themes tend to be short, they may allow interesting indexing schemes to be applied.

Quality and speed are two interdependent factors in evaluating melodic search algorithms. Of course the ideal solution will increase both quality and speed. Knowing about the extremes along each axis is a prerequisite for algorithm evaluation. Known algorithms such as indexing can be very fast even in a large database, but our work shows that fast algorithms that have been the focus of previous investigations are not delivering the best possible precision.

Our study of melodic similarity has led to an interesting improvement in which edit distance calculations implicitly constrain tempo variations, and this algorithm outperforms all others we have tried. Observing that frame-based approaches tend to work better than event-based approaches, we can consider further enhancements such as a probabilistic treatment of edit cost functions. Although slow, our frame-based algorithm can serve as a benchmark against which other algorithms can be compared.

7. FUTURE WORK

In the future, we plan to incorporate a probabilistic model of pitch estimation into the frame-based approach. By measuring actual pitch estimation errors from hand-labeled queries, we should be able to improve our edit cost functions. Similar training was used in another melodic similarity task [6].

Optimization of frame-based matching is important for practical applications. A frame-based algorithm could be the final pass after a faster query narrowed the database to a small set of candidates. Alternatively, some sort of frame-based algorithm might be run at lower time resolution to increase speed. Tempo estimation and key estimation might reduce the need to search over different tempos and keys. Finally, it might be possible to build a database index based on melodic fragments (shorter than themes) matched using frame-based techniques.

8. ACKNOWLEDGEMENTS

Dominic Mazzoni implemented the pitch analysis and transcription software as well as the initial version of the frame-based algorithm. Ann Lewis assisted with data collection and algorithm evaluation. This work is a part of the MUSART project (http://musen.engin.umich.edu/ musearts.html) and has benefited greatly from conversations with other project members. This work was supported by the National Science Foundation, Award #0085945.

9. REFERENCES

[1] Birmingham, W.P., Dannenberg, R.B., Wakefield, G.H., Bartsch, M., Bykowski, D., Mazzoni, D., Meek, C., Mellody, M. and Rand, W., MUSART: Music Retrieval Via Aural Queries. in *International Symposium on Music Information Retrieval*, (Bloomington, Indiana, 2001), 73-81.

[2] Blackburn, S. and DeRoure, D., A Tool for Content Based Navigation of Music. in *ACM Multimedia '98*, (Bristol, UK, 1998), ACM, 361-368.

[3] Dannenberg, R.B., An On-Line Algorithm for Real-Time Accompaniment. in *Proceedings of the 1984 International Computer Music Conference*, (Paris, 1984), International Computer Music Association, 193-198.

[4] Durbin, R., Eddy, S., Krogh, A. and Mitchison, G. *Biological sequence analysis*. Cambridge University Press, 1998.

[5] Ghias, A., Logan, J., Chamberlin, D. and Smith, B.C., Query by humming - musical information retrieval in an audio database. in *Proceedings of ACM Multimedia 95*, (1995).

[6] Grubb, L. and Dannenberg, R.B., A Stochastic Method of Tracking a Vocal Performer. in *1997 International Computer Music Conference*, (1997), International Computer Music Association.

[7] Hewlett, W. and Selfridge-Field, E. (eds.). *Melodic Similarity: Concepts, Procedures, and Applications*. MIT Press, Cambridge, 1998.

[8] Itakura, F. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-23*. 52-72.

[9] Keogh, E.J. and Pazzani, M.J., Derivative Dynamic Time Warping. in *First SIAM International Conference on Data Mining*, (Chicago, IL, 2001).

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

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

[12] Meek, C. and Birmingham, W.P., Thematic Extractor. in *2nd Annual International Symposium on Music Information Retrieval*, (Bloomington, Indiana, 2001), Indiana University, 119-128.

[13] Mongeau, M. and Sankoff, D. Comparison of Musical Sequences. in Hewlett, W. and Selfridge-Field, E. eds. *Melodic Similarity Concepts, Procedures, and Applications*, MIT Press, Cambridge, 1990.

[14] Nishimura, T., Hashiguchi, H., Takita, J., Zhang, J.X., Goto, M. and Oka, R., Music Signal Spotting Retrieval by a Humming Query Using Start Frame Feature Dependent Continuous Dynamic Programming. in *International Symposium on Music Information Retrieval*, (Bloomington, Indiana, 2001), 211-218.

[15] Rabiner, L. and Juang, B.-H. *Fundamentals of Speech Recognition*. Prentice Hall Signal, Englewood Cliffs, NJ, 1993.

[16] Rolland, P.-Y. and Ganascia, J.-G. Musical pattern extraction and similarity assessment. in Miranda, E. ed. *Readings in Music and Artificial Intelligence*, Harwood Academic Publishers, 2000, 115-144.

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

[18] Selfridge-Field, E. Conceptual and Representational Issues in Melodic Comparison. in Hewlett, W. and Selfridge-Field, E. eds. *Melodic Similarity: Concepts, Procedures, and Applications*, MIT Press, Cambridge, MA, 1998.

[19] Tolonen, T. and Karjalainen, M. A computationally efficient multi-pitch analysis model. *IEEE Transactions on Speech and Audio Processing*, *8* (6).