Approximate Matching Algorithms for Music Information Retrieval Using Vocal Input

Richard L. Kline
Computer Science Department
Pace University
New York, NY USA
rkline@pace.edu

Ephraim P. Glinert
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY USA
glinert@cs.rpi.edu

ABSTRACT

Effective use of multimedia collections requires efficient and intuitive methods of searching and browsing. This work considers databases which store music and explores how these may best be searched by providing input queries in some musical form. For the average person, humming several notes of the desired melody is the most straightforward method for providing this input, but such input is very likely to contain several errors. Previously proposed implementations of so-called *query-by-humming* systems are effective only when the number of input errors is small. We conducted experiments which revealed that the expected error rate for user queries is much higher than existing algorithms can tolerate. We then developed algorithms based on approximate matching techniques which deliver much improved results when comparing error-filled vocal user queries against a music collection.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Audio input/output*; H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*Methodologies and techniques*

General Terms

Human Factors, Algorithms

Keywords

query by humming, music information retrieval

1. INTRODUCTION

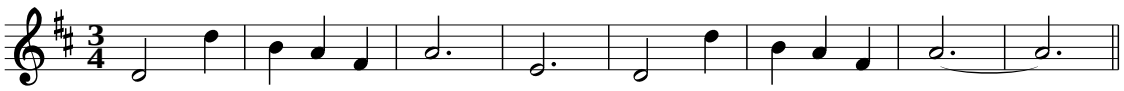
Computers have afforded ever-improving methods for the storage, indexing, and searching of many traditional media such as the books and journals found in libraries. The earliest computerized catalog systems were designed to mimic physical card catalogs, creating the immediate benefits of eliminating the manual labor and redundancy involved in maintaining three sets of cards ordered by subject, by author, and by title. As catalog systems grew in sophistication, they allowed more complex query formulations and expanded the number of searchable fields. Today, almost all publishing is done on computers. With continued advances in technology and nearly limitless data storage capacity, detailed indices of journals grouped by field of interest are now available online. Abstracts of journal articles, and often even their full text with figures, can be viewed directly from the computer screen. The power of these modern systems is of unquestionable benefit to the researcher seeking information, often reducing the task of searching for relevant published information from days of manually browsing printed materials to minutes of computerized searching.

The same advances in processing power and data storage capacity have opened up computing to many new areas of multimedia. Scores of novel applications have exploded into both consumer and commercial markets, giving casual users and professionals alike access to enormous collections of audio and video data through CDs and DVDs, commercial service providers, and public web sites. However, the usefulness of this wealth of information will be largely determined by how effectively these collections can be manipulated and searched. In stark contrast to the powerful text-based systems described above, most current multimedia databases lack both automatic indexing techniques and suitable representations in which queries can be formulated naturally.

As with the text databases of today, the interface to a media search engine requires careful design. The user must have sufficient control over the search process to produce results that match his or her intentions, yet avoid returning an overwhelming number of 'close' but undesired matches. Perhaps most significantly, the user must be able to formulate queries in a language appropriate to the media being searched. Since the most appropriate methods of input may be prone to errors both by the user and by the interface which processes the input, these errors must be anticipated and corrected where possible. In this paper we tackle issues such as these with respect to music, both to give focus to our efforts and because of music's importance as a pervasive element of human culture.

As a result of our continual exposure to music from a young age, we commonly possess the ability to remember and recognize thousands of melodies and tunes. Although most people are not sufficiently proficient with a musical instrument to be able to reproduce a given piece of music, they can sing, hum, or whistle. Even so, those who have not had formal music training often have difficulty accurately reproducing a melody by these vocal means. They tend not to be in key, and they often produce rhythms and pitch intervals incorrectly. Nevertheless, many people can hum a tune well enough so that other human listeners who are familiar with it can recognize it, as imperfect as the rendering may be.

MM'03, November 2–8, 2003, Berkeley, California, USA.



```
PC:       +    -  -  -    +    -       -     +    -  -  -   +
4-gram:   +---    --+-    +--+         -+--       ---+
               ---+  -+--       --+-          +---
index:     54    2  6 18   56    6      18    54    2
```

Legend:

`PC`	pitch contour values
`4-gram`	n-grams of length four (displayed in two rows so they can be shown beneath the PC symbols which begin them)
`index`	numeric representation of the n-gram, calculated by converting the contour to a base-3 number using (+ = 2, ˜ = 1, - = 0)

Figure 1: Encoding of a musical phrase into an n-gram representation.

The focus of our work is to ensure that a music information retrieval (MIR) system provides enough tolerance of errors that it can successfully identify a song by processing a query which is hummed both by people who have strong vocal skills and by those who are more melodically challenged. While the music collection can be expected to be accurate, the unpolished rendition of a tune as hummed or sung by the nonprofessional may appear quite different from the intended song. The performance key chosen may be arbitrary and may change one or more times during the course of the input. The time signature and tempo will not be reported. The durations, relative pitch values, and amplitudes of the individual hummed notes will be rendered approximately and may not be consistent throughout the phrase. How, then, shall we reconcile these two very different representations of music?

2. EXPERIMENTS IN HUMAN HUMMING

In order to ensure that a system relying on hummed user input would perform well for the average person, we conducted a study of human humming skills to augment and extend the results of previous studies in music perception, recognition, and reproduction, *e.g.*, [4, 8, 13, 14]. We quantified the nature and frequency of errors typically introduced into vocal renditions of familiar and unfamiliar tunes, as well as the differences in performance between those with musical training and those without. The results of this study formed the basis of a series of algorithms designed to match an input query to its intended song stored in a database of music.

Fifteen unpaid volunteers, nine male and six female participated in a single 30-minute session. They hummed a total of 19 popular folk tunes and a sequence of 32 five-note melodic fragments adapted from a similar but smaller study by Lindsay [16]. The subjects were divided into three groups based on their level of formal and informal musical training. The musician (MUSI) group comprised five people whose mean experience was 14.8 years, though only one had significant vocal experience and none had any voice training. The non-musician (NONM) group had four subjects with less than two years' experience, and the remaining six subjects formed an in-between group (BTWN) with a mean of 3.2 years of musical experience.

Several significant findings from this study influenced our algorithm development. For example, it was seen that while subjects were not good at holding notes for the intended duration, the time between the start of one note and the start of the following note was much more was much closer to a regular rhythm. We used this inter-note onset time (INOT) in most of our subsequent efforts. We also observed that subjects usually enlarged the smallest pitch intervals (of one or two semitones) between consecutive notes but compressed larger intervals (those greater than five semitones), and that these effects were more pronounced for rising intervals than falling ones. Perhaps the most important result was the number of errors found in the input: between subjects' mistakes and errors introduced by the `Autoscore` [24] music transcription software used, an average of 20% of the notes in any given input were found to be extraneous or missing. A full discussion of the study and its results is beyond the scope of this paper.

In addition to the valuable information obtained from analysis of the data collected in this study, it also provided us with a set of 172 sample query strings which were utilized in the testing and development of our search algorithms described here.

3. RELATED WORK

Several groups have performed relevant work in searching music databases, especially with hummed input. We describe here two categories of approaches to the problem.

3.1 Fast Matching Using Contours

One method of representing a melody is to remove all information regarding note durations to focus solely on pitch values. Groups such as Ghias *et al*. [6], McNab *et al*. [18], and Uitdenbogerd and Zobel [23] examined the usefulness of *pitch contours* when performing query searches. A pitch contour considers pairs of consecutive pitch values throughout a musical phrase and records only whether the second note of the pair is higher, lower, or remains at the same pitch value as the first note. This representation was thought to be helpful for a query-by-humming system in dealing with the pitch and duration inaccuracies bound to be present in hummed input queries. A grouping method known as *n-gram representations* was employed by Uitdenbogerd and Zobel and by Downie and Nelson [5] as a means of providing more efficient searches using these representations. Each group of symbols is converted to a single index number which can be looked up quickly. The transformation of a sample musical phrase into pitch contour data and then to n-grams of length four is shown in Figure 1.

We independently developed the n-gram grouping technique in our own early work and tested it on our own query set using the pitch contour representation. As we expected, the matching results obtained by this algorithm were extremely poor for all but the best few input queries. We went on to develop 24 other algorithms and variants which utilized the ideas of contours and the fast matching of n-grams. None of these gave acceptable results for anything larger than a toy-sized database of only a few dozen songs. A thorough discussion of the methods developed and tested is beyond the scope of this paper.

3.2 Approximate Pattern Matching

An approximate string or pattern matching algorithm compares two sequences of data and calculates their similarity by way of an *edit distance*. A set of rules specific to the data being compared defines the costs of operations which are used to transform one sequence to the other, and the total of these costs is given as the edit distance. Fundamental operations include inserting or deleting an element from one of the sequences, and transforming one individual element into another. As a simple example, in comparing two character sequences `abbc` and `abcd`, the former can be transformed to the latter by many sequences of steps; two such transformations are shown in Figure 2. The algorithm identifies the transformation sequence yielding the smallest edit distance, reporting that value as the matching score for the pair.

`abbc` → `abc` → `abcd` (deletion, insertion)

`abbc` → `abcc` → `abcd` (transform, transform)

Figure 2: Two possible sequences to transform one character sequence to another.

3.2.1 Method

To calculate the similarity between two sequences A and B with lengths a and b respectively, a matrix M of size $(a+1) \times (b+1)$ is created. The individual elements are calculated according to the following recurrence relation:

$$M[i,j] = max \begin{cases} M[i-1,j] - d_1 \\ M[i,j-1] - d_2 \\ M[i-1,j-1] + t \\ 0 \end{cases} \quad (1)$$

In this equation, d_1 and d_2 represent the cost of inserting a gap into each of the two sequences, respectively, and t represents the cost of transforming element A_i to B_j. Frequently d_1 and d_2 do not need to be distinct and are replaced by a single gap penalty d, which may be a simple constant value or a function dependent on the number of consecutive gaps inserted. Similarly, t can be a constant, though it is normally determined by a function or a look-up table.

If the two sequences are being compared in their entirety, the elements in row 0 and column 0 must be calculated using only the portion of the formula representing insertion of gaps. When the algorithm is used instead to find the most closely matching *subsequences* of A and B, these spaces are filled with zeros before processing begins.

Once the first row and column are filled in as appropriate for the type of matching being performed, the remaining values for the entire matrix are calculated. The highest resulting score in the matrix represents the minimal edit distance between the two sequences. To find the corresponding subsequences representing this score, the matrix values can be traced back from the maximal value in the opposite manner of the way they were calculated.

3.2.2 Previous Applications

Smith and Waterman [22] were among the first to apply approximate matching algorithms to the comparison of biological molecular sequences. The method they developed has been used extensively to compare structural similarities between pairs of DNA and also between pairs of proteins. DNA is composed of long sequences of 4 nucleotides, while proteins are made up of sequences of 20 amino acids. The values for t which describe the similarity of two given symbols are provided in a table based on heuristics estimating the relationships among the bases (for DNA) or acids (for proteins). The enormous length of sequences representing DNA and proteins makes this process very expensive computationally. Subsequent work in bioinformatics has focused on preserving the matching power of this algorithm while reducing the computation time required. Notable improvements include the BLAST family of algorithms by Altschul *et al*. [1, 2], which rely on heuristics to improve search times.

Mongeau and Sankoff [19] were among the first to adapt these techniques to the comparison of musical pieces. They compared pitch intervals between two musical sequences, defining t as a look-up table based on the difference between intervals; musically harmonious intervals such as a fifth (seven semitones) were given very low transformation values, while dissonant intervals were assigned higher costs. Mongeau and Sankoff also introduced additional calculations to weight separately the notion of a single note matching more closely to a set of notes in the corresponding sequence. They used the algorithm to compare the similarity of entire pieces of music and the styles of disparate composers.

McNab *et al*. incorporated these ideas into their `MELDEX` system [18] while comparing a relatively short user query string to entire songs. However, the relative complexity of this algorithm led them also to include a faster but less precise method devised by Wu and Manber [25] as their default search method.

Uitdenbogerd and Zobel [23] examined the effectiveness of an approximate pattern matching algorithm comparing songs represented by pitch intervals given in semitones, finding it to be the most accurate of the matching methods they tested. They used $d = -2$ and had only two possible values for t: $t = 1$ if the intervals matched, and $t = -1$ if they did not. This scoring method was feasible in their work, which assumed error-free query strings, but is not appropriate for imprecise hummed input.

Most recently, Pauws [20] used a more complex recurrence relation along the lines of Mongeau and Sankoff in considering several different transformation possibilities when calculating each matrix element, additionally weighting the scoring using duration information from the notes involved.

Some have suggested other techniques such as the use of Hidden Markov Models to attack the matching problem. However, they would likely be much less tolerant of any errors in the individual music database songs than the types of approximate pattern matching described here. While our own system used a database without errors, difficulties in transcribing music from recordings or even sheet music leave open the possibility of errors in the song collection.

4. APPROXIMATE MATCHING USING PITCH INTERVALS

As we saw in our own experimentation in developing contour-based matching algorithms, pitch-based contours proved to be more discriminating than information based on note duration information. This observation led first to the idea of using only pitch information in the design of an approximate matching algorithm.

In our view, the most straightforward application of approximate string matching to the problem of music queries was to define similarity with respect to pitch intervals. While others have proposed and tested matching algorithms based solely on pitch data as described above, none have described the approach we have taken in utilizing pitch interval values directly, without additional transformation into other representations.

4.1 Design

Both user queries and database entries were converted into sequences of pitch interval values given in semitones. For the comparison of arbitrary elements of the two sequences A_i and B_j, we assign a simple pairwise arithmetic difference calculation:

$$t \;=\; C - |A_i - B_j| \qquad (2)$$

We treated C as the maximum difference between pitch intervals to be considered as making a positive contribution to the localized matching score. With an eye toward computational efficiency, we used constant values for C in all of our algorithms developed using this approach. The gap penalty function d was also implemented as a constant value, giving a uniform linear penalty based only on the size of an introduced sequence gap which takes almost no time to compute. Tests were run on the full set of user input queries for several combinations, ranging from 3–6 for C and 0.5–2.0 for d, to empirically find the optimal values. The algorithm is referred to as PI-S (for Pitch Intervals, Solo measure).

After calculating the similarity matrix, the resulting score was normalized by dividing it by the square root of the length of the shorter of the two strings being compared. The PI-S algorithm, and many of the others described in this paper, were tested both with and without this normalization step. In each case where it was used, it produced consistently better results.

4.2 Results

Several different combinations of values for C and d produced similar overall results, with slight variations in the matching scores of some input trials, but with no evident pattern to explain the differences. The combination of $C = 4$ and $d = 1$ were chosen empirically as the best values among those we tested; the results reported here were generated using those values.

The results for this and subsequent algorithms are given in several measures. The **Rank 1** designation reports how many times the intended song represented by a given input query was correctly identified as the single best match in the music database. **Rank 1–10** indicates the intended song scored among the top ten results returned by the algorithm, while **Top 2%** counts how often the intended song scored among the highest two percent of database songs. For tests performed against the full test database, this corresponds to a ranking in the range 1–71. The database used in our testing was the Digital Tradition Folksong Database [7], a collection of nearly 3,600 songs freely available on the internet.

The matching scores obtained with PI-S were a dramatic improvement over all of the results obtained using only contour information. As can be seen in Table 1, the algorithm correctly matched the intended song more than 80% of the time for four of the subjects, and 50% overall. In 61% of the cases, the correct song appeared in the top ten results. When results were grouped by song, two of the test songs (*Twinkle Twinkle Little Star* and *Happy Birthday*) stood out, recognized within the top ten results for 13 of 15 subjects.

At the other end of the spectrum, none of the queries were recognized correctly for two of the subjects, and two of the songs tested were identified within the top ten results for as few as 33% of the subjects. Nonmusicians as a group hummed well enough for the intended song to be ranked number one only 38% of the time, and in the top ten results 52% of the time.

PI-S: results by subject				
Subject/ Group	**Rank 1**	**Rank 1–10**	**Top 2%**	**Total Trials**
1	11	11	11	12
2	7	9	9	12
3	7	8	9	10
12	2	3	7	11
14	9	11	12	12
MUSI	63%	74%	84%	57
7	0	1	3	12
8	6	7	7	10
9	8	10	10	12
11	4	5	7	11
13	5	6	10	11
15	9	9	10	11
BTWN	48%	57%	70%	67
4	4	7	9	12
5	10	11	12	12
6	4	7	7	12
10	0	0	2	12
NONM	38%	52%	63%	48
ALL	50%	61%	73%	172

Table 1: Results of PI-S matching algorithm, tabulated by subject.

4.3 Discussion

For the matching algorithm designed by Mongeau and Sankoff and implemented in `MELDEX`, intervals such as one semitone were given very high cost values, yet in our own experiments it was observed that subjects attempting to hum the same note twice often ended up producing notes which were transcribed to be one semitone apart. Mongeau and Sankoff chose the values which comprise their transformation matrix t based on tonal similarities as described in music theory. Their pitch distance measure is far less appropriate for use with the imprecise input strings obtained from hummed renditions of music due to the tendency of users to produce unintentionally dissonant pitch intervals.

Similarly, the algorithm used by Uitdenbogerd and Zobel relied on query input which was free from errors, so the simple binary values of match/no match made sense in their application area but is quite impractical in representing hummed input.

The extra step of normalizing the resulting similarity score by the lengths of the two compared sequences improved the matching results noticeably. This added calculation was valuable because of the large disparity in the lengths of songs found in the database. This technique was incorporated into most of the subsequent algorithms we developed, including those described here.

While PI-S produced excellent results in a number of cases, there were other songs and subjects for which the matching scores were quite poor. The robust matching offered by approximate pattern matching provided huge gains over contour-based algorithms when tested against our body of error-filled input samples.

Overall, though, there was room for improvement, particularly for those subjects and songs which proved difficult to match. Based on the promising idea of combining pitch and duration information, we set about devising and testing algorithms which incorporated

additional features from the input data. We wanted to represent duration information in a more abstract manner than in the raw timing data given in milliseconds by the transcription subsystem.

5. TEMPO ESTIMATION

The output of the music transcription process includes note timings given in absolute milliseconds from the start of the input. Only a small percentage of hypothetical users of an MIR system could be expected to consciously quantify a priori the tempo at which they intended to hum an input phrase. A transcription program cannot be expected to identify a nonexistent tempo. On the other hand, anecdotal observation of our subjects and informal study of the resulting input data indicated that a tempo estimate could be inferred from these samples.

Several researchers have explored the process of extracting the tempo or beats directly from an acoustic recording, among them Dannenberg (*e.g.*, [3]) and Scheirer [21]. Many of these projects were designed for real-time processing, to be applied to applications such as automated accompaniment of live performances.

The requirements of an MIR system when processing hummed input were less involved than those of the other researchers mentioned, and the `Autoscore` music transcription software took care of the task of identifying individual notes out of a single-note acoustic input stream. Furthermore, a tempo estimation algorithm suitable for use with hummed input needed to be able to make a good guess at a tempo for queries of any length. Very short input phrases had to be expected, producing relatively little data on which to base an estimate. Our algorithm would need to be able to return a reasonable result whether it was given five notes or 25 from which to approximate the value of the performance tempo.

The data received from the pitch transcription software for the purpose of inferring a tempo value from an input phrase include the time each note begins, measured from the beginning of the recording, along with the INOT duration of the note. As mentioned in the description of our experimental studies, it was shown that the inter-note onset time (INOT) was a more reliable measure of a user's intended note durations than the actual duration values reported by the transcription software for each note.

5.1 Method

If most of the INOT values were clustered around a single value, one might be tempted to use a simple least-squares difference measure to find the best point to represent the cluster. However, the least-squares computation is particularly susceptible to distortion in the presence of outlier points. One popular robust technique known as M-estimators [15] provides a method to evaluate closeness of fit while minimizing the effect of outliers.

For almost any song, INOT values generally cluster around two or possibly three points. Furthermore, because they are based on the rhythm and timing of the song, these cluster points normally have a simple arithmetic relationship of 1:2, 1:3, or 1:4. In the case of a relatively short input query, it is more likely that only one or at most two such points can be identified. Based on informal analysis of the INOT values of the input queries we collected, we designed the algorithm to expect the majority of notes to fall into two clusters in the ratio of 1:2.

Thus, we have defined our own algorithm in the style of an M-estimator but with the added property that data is fit to the parameter being estimated at two related points rather than one.

The cost of a parameter estimate candidate c is defined to be

$$Cost(c) = \sum_{p} r(d(p,c)) \tag{3}$$

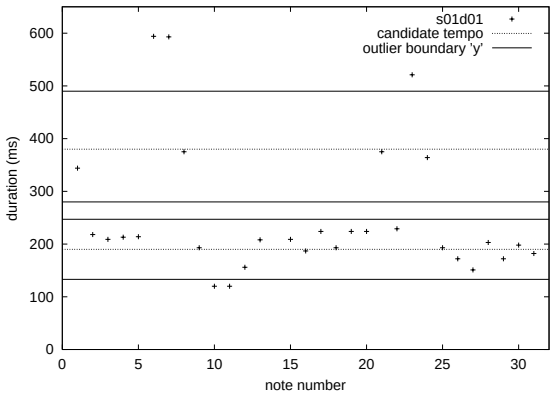


Figure 3: Plot of duration values for a user's hummed input with candidate tempo c and outlier boundaries given by y.

for all points p in the data set. Given a point p and a cluster candidate c, we define distance as

$$d(p,c) = min\frac{|p-c|}{c}, \frac{|p-2c|}{2c} \tag{4}$$

The contribution to the $Cost$ function for p is based on the value d but is governed by another test:

$$r(d) = \begin{cases} d, d < y \\ 0, d >= y \end{cases} \tag{5}$$

The calculated distance to the nearest tempo candidate is scaled by the value of that candidate, so that notes being measured against the $2c$ cluster point contribute d values in the same proportion as those measured against the c candidate value. The parameter y determines the maximum distance an INOT value may be from its nearest cluster point (c or $2c$) and still contribute to the total $Cost$. Notes outside of this range are rejected by the formula for r but instead are counted separately as outliers. This definition of r is the mechanism that prevents those few notes whose INOT values are much higher or lower than average from skewing the tempo estimation calculations; were they to be considered, they would add a large d value to the overall $Cost$.

One observation during the initial testing of this algorithm was that it would sometimes choose a tempo based on a low $Cost$ score computed by classifying a significant number of notes as outliers. To counteract this undesired result, we added an extra provision limiting the total percentage of outlier values allowed if a candidate tempo value was to be considered. If this cutoff causes every candidate tempo to be rejected because too many of the notes end up being classified as outliers, the threshold is increased iteratively until at least one candidate tempo produces a sufficiently low outlier count. Another check ensures that, for a given candidate tempo, a sufficient percentage of the notes in the song are being compared against the base c value. This additional step eliminates the case where the algorithm otherwise finds a lower $Cost$ by incorrectly selecting the candidate tempo that should really be the $(2c)$ value, pushing most or all of the notes which are near the intended c value down into the outlier range. The example given in Figure 3 should make this description more clear.

We anticipated that songs written in other time signatures such as $\frac{3}{4}$ or $\frac{6}{8}$, where long notes would be expected to be held three times as long as the base note duration, would cause this algorithm to fail to find the correct tempo. However, when we applied the algorithm

to those samples in our collection and from our subjects which fell into this category, we were pleased to discover that the algorithm correctly identified the tempo as accurately for these songs as it did for songs which used $\frac{2}{4}$ or $\frac{4}{4}$ time signatures. We believe the main reason for this surprising and beneficial behavior is that, regardless of the time signature of the song, notes having durations of approximately one beat will tend to outnumber other duration values, and our algorithm normally finds that dominant value. The notes whose durations are three times that of the dominant duration contribute larger distance values when calculated against the $(2c)$ tempo candidate than they would if compared to a $(3c)$ candidate, but their relative rarity diminishes the negative effect they have on the overall distance measure, and in fact some of them end up as outliers, contributing nothing to the total.

There are cases where the tempo value reported by our algorithm is incorrect. The most interesting situation occurs when a given song has multiple sections where the tempo changes from one section to another. Our algorithm only attempts to identify a single tempo for any given song, so the song is encoded according to whichever tempo value dominates the file according to its evaluation function. On the other hand, a user might hum as a query a portion of the song from a section whose tempo differs from the one calculated for the database representation, with the resulting mismatch impeding efforts to identify the song correctly. A similar source of potential problems crops up when a song is characterized by a single tempo throughout, yet it has one or more sections for which longer notes (perhaps half notes or whole notes) dominate the melody. If the user hums an input phrase belonging exclusively to one of these slower sections, it is likely that the tempo estimator will identify these longer notes in the user query as the base value for tempo determination purposes. Again, the interpretation of the user input in light of this tempo estimate will differ from the corresponding section of the intended match from the database.

One potential solution to this problem is to store two separate encoded versions of each song in the database: once using the original calculated tempo, and a second time at half that value. This approach was used in the `SoundCompass` system implemented by Kosugi *et al*. [12], which also made use of beat-based data representations.

5.2 Discussion

Two other research groups have included features in their systems which incorporate tempo-based information in their search algorithms. The approach taken in `SoundCompass` [12] is to require users of their program to set a virtual metronome to their intended tempo before they begin humming, doing their best to keep their humming in time with it. The `MELDEX` system [17] offers "rhythm matching" as an advanced option in their system; when this feature is selected, the user is required to explicitly specify the tempo at which the input query will be evaluated along with the minimum beat-based duration value it should recognize (*e.g.*, quarter note, eighth note, etc.).

Our tempo estimation algorithm provides this same information without burdening the user with the task of tempo estimation, which may not be obvious or easy to someone without musical training.

6. DURATION-WEIGHTED PITCH INTERVALS

With an effective tempo estimation algorithm in hand, we developed additional algorithms designed to augment the matching power of pitch interval data by incorporating duration information into the scoring method.

6.1 Design

The design of this algorithm attempted to address several perceived problems from the initial pitch-only PI-S design. First, we wanted to minimize the negative effects of extraneous notes (both those erroneously injected into the data files by the transcription process and those introduced by the user) on matching scores. Second, since longer notes are relatively rare in user input phrases, we wanted to emphasize the contribution made by such notes. Third, we wanted to avoid the situation where two compared notes scored as a good match because they had similar pitch intervals, even though one was much longer in duration than the other.

As an initial step, both user and database songs were converted to a representation where each note was given by a pair of numbers: the pitch interval in semitones as used in PI-S, and a duration value expressed not in milliseconds but in *beats*. A pitch interval is always calculated between two consecutive notes, so the duration value associated with a pitch interval was that of the first of the two notes involved in the transition.[1]

In this algorithm, named RePScaD (for Relative Pitch Scaled by Duration), the arithmetic difference between the respective pitch interval values is calculated as before. The resulting value is then *scaled* by the smaller of the two associated note durations expressed in beats. This scaling gives longer notes more weight in determining the score contributed by a given comparison. Very short notes have only a small chance to affect the overall score regardless of whether they were actually hummed or were introduced by transcription errors. By taking the smaller of the two durations, the case where a short note from sequence A is matched against a long note in sequence B has a smaller impact on the overall score, as opposed to two longer notes with the same relative pitch values. Since the dominant duration value for a user query is one beat by the nature of the tempo estimation algorithm, typical pairs of data being compared will have a scaling factor of one. The important features of the RePScaD algorithm can be expressed as follows:

$$\begin{aligned} s &= min(A_i.dur_{beats}, B_j.dur_{beats}) \\ t &= C - |A_i.pitch_{intvl} - B_j.pitch_{intvl}| \times s \end{aligned} \quad (6)$$

Along with most of the algorithms described here, C is treated as a constant and the gap penalty d is a constant function in this case. As before, we ran the algorithm using several values for C and d, and we report the best results obtained among these trials.

6.2 Results

The performance data summarized in Table 2 show that this algorithm did not perform as well as PI-S in any category of subject or group. There were individual trials for which it gave a better ranking of the correct song, but these were rare. The number of trials where the intended song was ranked first was 51% for MUSI, 28% for BTWN, and 17% for the NONM group.

6.3 Discussion

We cannot say with certainty why this algorithm gave such disappointing results as compared to the unscaled PI-S version, but there is a likely culprit. The largest contribution to a song's score occurs when long notes from query and candidate are aligned together. If the pitch interval values differ widely at these points, such

[1]As a consequence of there being only $n-1$ pitch intervals yet n duration values for a musical phrase of n notes, one of the duration values had to be discarded. It seemed more logical to associate with a pitch interval the duration value corresponding to the first note of the pair comprising the interval, thus the final duration value is the one dropped.

RePScaD: results by subject				
Subject/ Group	**Rank 1**	**Rank 1–10**	**Top 2%**	**Total Trials**
1	11	11	11	12
2	4	5	9	12
3	5	6	7	10
12	1	1	2	11
14	8	8	10	12
MUSI	51%	54%	68%	57
7	0	0	2	12
8	4	6	7	10
9	3	6	9	12
11	2	3	6	11
13	5	6	6	11
15	5	5	8	11
BTWN	28%	39%	57%	67
4	2	5	8	12
5	3	10	11	12
6	3	3	6	12
10	0	0	2	12
NONM	17%	38%	56%	48
ALL	33%	44%	60%	172

Table 2: Results of RePScaD matching algorithm.

pairings decrease the overall matching score substantially. This situation can occur any time an interval is preceded by an incorrect interval created from an erroneous note (whether the source of the error is the user or the transcriber).

Mongeau and Sankoff [19] introduced the idea of using duration information to weight pitch interval scores in their approximate matching algorithm. Their work focused exclusively on comparing written scores free of errors and irregularities in note durations, which may help to explain why their evaluation yielded better results than we obtained with error-filled query data. The MIR system built by Kageyama *et al*. [10] used some form of duration weighting in their matching algorithm as well, but the details of its implementation were not described.

7. RELATIVE PITCH AND RELATIVE DURATION INTERVALS

From the previous results it was clear that our attempts at incorporating duration information into a more robust matching algorithm were actually providing inferior results to the pitch-interval-only (PI-S) algorithm. After additional study of our user test data, we eventually devised a new technique for representing duration information, and from it we created our most effective matching algorithm. It is named RePReD (pronounced "rep-red") for Relative Pitch – Relative Duration.

7.1 Design

A review of the data transformations we had created and used to date revealed that, while our duration representations were free of the absolute values of milliseconds in favor of a beat-based representation, the fact that the values were scaled did not affect that this was still an absolute measure. What was missing was the duration equivalent of our pitch interval representation.

Early examination of the data collected in our study revealed that raw duration values generally were shorter than INOT values, and the discrepancy between the two increased as INOT increased. With the use of our tempo estimator, we examined the INOT data again after transforming it into beat-based values. With this additional information we confirmed a result that was somewhat evident from carefully listening to our user input: the longer the intended INOT duration measured in beats, the more likely the performed INOT would be compressed by a subject.

This observation led to our definition of *duration intervals*. As we had done with pitch intervals, we consider consecutive pairs of INOT values. For each pair, we divide the later value by the earlier one, then take the base-2 logarithm of the quotient. As an example, if a melody contained a one-beat note followed immediately by a four-beat note, the transition between them would be represented by the duration interval 2.0, *i.e*., $log_2(\frac{4}{1})$. If a subject attempted to hum these notes, he or she might end up with respective notes at 1.05 and 3.45 beats, which yields a duration interval of 1.72. The larger the difference between the pair of INOT values being considered, the more leeway is given in the log-based scale for the user to be off in his or her durations. Figure 4 provides an example of a short musical phrase transformed to relative pitch intervals and relative duration intervals.

After encoding all data files using this new representation, the RePReD algorithm compared user queries against the database using the following method of distance calculation:

$$\begin{aligned} t &= C - |A_i.pitch_{intvl} - B_j.pitch_{intvl}| \times s_p \\ &\quad - |A_i.dur_{intvl} - B_j.dur_{intvl}| \times s_d \end{aligned} \tag{7}$$

Here, the difference in the duration interval values contribute to the distance score alongside the difference in the pitch interval values. Each difference is in turn subtracted from a constant which we again varied in our testing along with the usual penalty gap constant d. The results shown are for the values $C = 5, d = 1$. As before, several values for C, d, s_p, and s_d were tested experimentally.

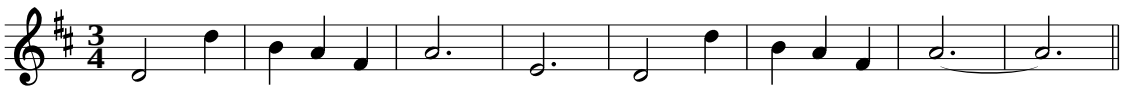
7.2 Results

As Table 3 demonstrates, the RePReD algorithm produces results superior to those using pitch information alone. When compared to the previous best algorithm, PI-S, the scores for the musician group improved by 5%, and the gains for the other groups were more significant, averaging 7% each across all categories. The average rank of all trials rose from 414 for PI-S to 248 here.

One of the main goals of this research has been to produce a system that would work well for as wide a variety of people as possible, regardless of their vocal skills. On the other hand, there will always be some people who simply cannot hum a recognizable tune. With that in mind, we also report for this algorithm the results for all but the three subjects who consistently gave the lowest search results among all algorithms tested. The **Best 12** category shows the average scores for all subjects once the three lowest scorers were removed. With this smaller set, the number of trials correctly identified at the first rank jumps to 68%, and the correct song is found within the top ten results 78% of the time. These numbers are nearly identical to those of the MUSI group.

7.3 Discussion

Many of the algorithms described earlier work well for those few people whose humming is of good quality and whose voices elicit few errors induced by the music transcription software. The results tables for most of these algorithms appear nearly bipolar, with certain subjects' input ranked near the top while others' were near the bottom. Throughout our work we considered the possibility that differences among people's humming skills would necessitate multiple search algorithms being used, with the system first inquiring about or testing a given user's abilities in order to select the



```
PV:    62   74  71 69 66   69      64    62   74   71 69 66    69
PINT:    +12  -3 -2 -3   +3  -5        -2  +12   -3 -2 -3    +3

DB:     2    1   1  1  1    3       3     2    1    1  1  1     6
DINT:   -1.0   0  0  0 +1.6   0      -0.6 -1.0    0  0  0  +2.6
```

Legend:

PV	MIDI pitch values
PINT	pitch intervals in semitones
DB	note durations, expressed in beats
DINT	duration intervals

Figure 4: Encoding of a musical phrase into relative pitch and relative duration intervals (RePReD).

RePReD: results by subject				
Subject/ Group	**Rank 1**	**Rank 1–10**	**Top 2%**	**Total Trials**
1	11	11	11	12
2	8	8	10	12
3	6	8	9	10
12	2	6	9	11
14	11	12	12	12
MUSI	67%	79%	89%	57
7	1	1	5	12
8	7	7	7	10
9	8	10	10	12
11	4	6	8	11
13	8	9	10	11
15	10	10	10	11
BTWN	57%	64%	75%	67
4	5	7	7	12
5	10	12	12	12
6	5	7	9	12
10	2	2	4	12
NONM	46%	58%	67%	48
ALL	57%	67%	77%	172
Best 12	68%	78%	84%	137

Table 3: Results of RePReD matching algorithm.

appropriate strategy for processing that user's hummed queries.

As was seen in this algorithm, however, it was possible to achieve dramatic improvements in the scores of non-musicians while at the same time upgrading the results of musicians as well. It was also surprising that the differences between the results obtained by the RePScaD and RePReD algorithms were as great as was observed. Indeed, it seemed counterintuitive that adding duration information – which seemed more reliable than pitch information from user queries – to a search would reduce the effectiveness of the search process, yet this behavior persisted through several iterations of algorithm design and testing, right up until the development of the RePReD algorithm.

In the `CubyHum` system, Pauws [20] independently developed a duration ratio and included it in his algorithm, though he did not include log-based scaling and his use of this information is more complex (and time-consuming to calculate) than our own.

8. EVALUATION OF *REPRED*

In order to discuss the effectiveness of the RePReD algorithm, we must consider how well the system performs as the size of the database being searched is increased. This factor impacts both the running time of a search and the quality of the results returned. We also compared its results against other current systems.

8.1 Computational Efficiency

In general, the dynamic programming approach to approximate pattern matching is not efficient. When comparing two sequences A and B with lengths a and b respectively, the cost of calculating one similarity matrix is $O(ab)$. When A is being compared to a database of n sequences, the total cost of finding the best match is $O(abn)$, assuming the values of b for the sequences stored in the database are comparable.

When working with data such as biological molecular sequences [22], a and b can reach into the millions, resulting a very slow algorithm. This computational expense is the reason for the development of heuristic improvements such as BLAST [1, 2]. These algorithms were specifically tuned to work with biological data, so while it is possible they could be utilized to improve the search efficiency in the MIR problem domain, some adaptation would be required.

In an MIR system using hummed input queries, unlike the situation for which Mongeau and Sankoff developed their dynamic programming algorithm [19], a user input query sequence A will have a small limit which can be considered constant. Similarly, even the longest recorded melody can get only into the range of a few thousand notes in the case of a symphony. In this way, a and b can both be considered constant values, reducing the effective complexity of our algorithms to $O(n)$, *i.e.,* linear in the size of the database.

The implementation described here compares an input query to a database of nearly 3,600 songs in less than one second on a 1.4GHz Athlon computer system. This speed is comparable to that of other current MIR systems, and the performance is more than fast enough for interactive use. Some potential applications of music information retrieval systems may require databases along the size of millions of melodies, but in most practical situations for which search results are expected to be returned in real time, the database size

Group	Rank 1		Rank 1–10		Top 2%	
	Small	Full	Small	Full	Small	Full
MUSI	78%	67%	89%	79%	89%	89%
BTWN	62%	57%	74%	64%	74%	75%
NONM	56%	46%	70%	58%	70%	67%
ALL	67%	57%	78%	67%	78%	77%
Best 12	76%	68%	86%	78%	86%	84%

Table 4: Comparative search results between the average of 360-song subsets and the full 3,600-song database for RePReD.

can be limited through the use of metadata. For instance, one likely application of this research is to provide a search mechanism to be used at a music store or library. In such a situation, a user of the system often will be able to specify at least one additional piece of information, such as the genre of the music, or perhaps the artist, publisher, or a few words of the lyrics of the song. The inclusion of such metadata can very quickly reduce the search space into the thousands, if not hundreds, of database entries.

8.2 Scalability of Results

In order to test how well the RePReD algorithm identifies the correct song when the size of the database changes, we created several random subsets of approximately 360 songs out of our full 3,600-song database. All of the user input queries were tested against the subsets using the RePReD algorithm, and the search results for each of these smaller databases were recorded and averaged.

These results are detailed in Table 4 and shown alongside the corresponding results from using the full database (reproduced from Table 3). Not surprisingly, the correct song was identified as the top-ranked result or within the top ten results less often when the database was larger, as could be expected. For all subjects, instances where the correct answer was ranked first diminished from 67% to 57% when the database grew by a factor of ten; similarly, the correct answer appeared in the top ten results 78% of the time for the smaller databases, but in 67% of the trials involving the full database. When considering how often the rank of the correct answer falls within the top 2% of the database, however, there is only a 1% dropoff as the database grows by a factor of ten.

8.3 Comparative Results

Several other systems implementing query-by-humming have reported success rates very similar to our own. For a better comparison against our RePReD algorithm, we directly tested several of these other systems against our own using identical input queries.

The `SoundCompass` system developed by Kosugi *et al*. and reported in [12] gave search results within one second on a dual-processor Pentium III 500MHz system against a database of approximately 11,000 songs. They tested 183 user queries, and their best algorithms ranked the correct song first about 70% of the time and within the top ten 78% of the time. These numbers are very close to both our MUSI group and the average of the top 12 of our 15 subjects as shown in Table 3. `SoundCompass` is not currently available for public testing, but Kosugi kindly shared with us sixteen hummed samples of *Jingle Bells* recorded by their group during testing. [11] For these samples, the latest version of their system (which had grown to 20,000 database songs) ranked the correct song first for five of the queries, and within the top 100 for nine of the 16. We ran the same samples through our own database: the RePReD algorithm ranked the correct song first in 15 of 16 samples, with the final one ranked third. For most of these queries, the score of the correct match in the database was an extremely significant 10–20% higher than the second-ranked song.

Jang *et al*. [9] also reported test results very similar to both `SoundCompass` and to our own work for their `SuperMBox` system: for 1,000 input samples matching only against the start of a melody in their 3,000-song database, the correct song was ranked first 59% of the time and in the top ten 71% of the time. Their system took nearly the same time as ours for similarly-sized databases when set to search only the start of a melody, while allowing a search to match anywhere took much longer – about 30 seconds, and they did not report matching results for this condition. We tested `SuperMBox` with 30 of our input samples taken from six subjects whose input quality varied widely. When matching only at the start of a melody, `SuperMBox` ranked the correct song first in 10 of 30 cases, vs. 16 of 30 for RePReD. When matching anywhere within a song, `SuperMBox` ranked the correct song first in only six of 30 queries.

We also attempted to compare our results to those provided by the New Zealand Digital Library `MELDEX` system [18]. We sent ten of our better input samples to their web-based query system, which contained 9,000 songs, including all of those in our own test database. `MELDEX` returned *zero* matches for six of these samples, and in three of the four remaining, the intended song was not in the set of returned results. Only in one case did it correctly identify the song as the top-ranked result.

Several factors prevent us from drawing any broad conclusions from these comparisons. The databases differed in composition and in size by more than a factor of six. `SoundCompass` queries must be hummed in time with a metronome and must be at least four measures in length. The number of samples tested here was quite small. Nevertheless, these comparisons are very encouraging in showing the powerful matching ability of the RePReD algorithm with error-filled input.

9. CONCLUSION

Approximate pattern matching offers a robust method of performing inexact comparison of strings. A straightforward yet novel application of this technique to an algorithm matching music only through pitch interval data (PI-S) against our collection of real-world hummed input queries gave good results for more skilled subjects but worked poorly for those with lesser humming abilities.

Several new algorithms were developed to incorporate duration information into the similarity calculations. These new matching algorithms made use of data in ways suggested by the analysis of our experimental studies on the nature and frequency of errors in music hummed by untrained subjects. To support this goal, an algorithm was developed to find a reasonable estimate of the tempo of a user's hummed query. Most of the subsequent algorithms developed proved less effective than the pitch-only PI-S method.

The resulting RePReD algorithm gave results which improved upon those of the PI-S algorithm, most dramatically for those subjects whose hummed input had consistently been hardest to identify. The algorithm makes better use of both pitch and duration information than previous efforts. RePReD correctly identified a hummed tune out of a set of almost 3,600 songs for most subjects over two-thirds of the time, and it placed the correct song within the top ten results for almost 80% of our test trials. Direct comparisons with other existing MIR systems suggest that RePReD is better at identifying user queries when several input errors, typical of users with relatively little music experience or ability, are present.

10. ACKNOWLEDGEMENTS

This research was supported in part by the National Science Foundation under contracts EIA-9214887, EIA-9214892, IIS-9213823, CCR-9527151, and EIA-9634485. It represents a portion of the dissertation research of the first author, performed under the guidance of the second author.

The authors gratefully acknowledge Naoko Kosugi of NTT Laboratories for providing a set of sample queries from his group's research studies.

11. REFERENCES

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *J. Mol. Biol.*, 215:403–410, 1990.

[2] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[3] R. B. Dannenberg. An on-line algorithm for real-time accompaniment. In *Proc. 1984 International Computer Music Conference (ICMC '84)*, pages 193–198, 1984.

[4] W. J. Dowling. Scale and contour: Two components of a theory of memory for melodies. *Psychological Review*, 85(4):341–354, 1978.

[5] J. S. Downie and M. Nelson. Evaluation of a simple and effective music information retrieval method. In *Proc. 23rd ACM Conf. on Research and Development in Information Retrieval (SIGIR '00)*, pages 73–80, Athens, Greece, 2000.

[6] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query by humming: Musical information retrieval in an audio database. In *Proc. 3rd ACM Conf. on Multimedia (MM'95)*, pages 231–236, San Francisco, Nov. 5–9 1995.

[7] D. Greenhaus et al. Digital tradition folksong database. Available (as of this writing) at **http://www.mudcat.org/folksearch.html**.

[8] A. R. Halpern. Memory for the absolute pitch of familiar songs. *Memory and Cognition*, 17(5):572–581, 1989.

[9] J.-S. R. Jang, H.-R. Lee, and M.-Y. Kao. Content-based music retrieval using linear scaling and branch-and-bound tree search. In *IEEE Int'l Conf. on Multimedia and Expo*, Tokyo, JP, August 2001.

[10] T. Kageyama, K. Mochizuki, and Y. Takashima. Melody retrieval with humming. In *Proc. 1993 International Computer Music Conference (ICMC'93)*, pages 349–351, Tokyo, 1993.

[11] N. Kosugi. Personal correspondence.

[12] N. Kosugi et al. A practical query-by-humming system for a large music database. In *Proc. 8th ACM Conf. on Multimedia (MM'00)*, pages 333–342, Marina del Rey, CA, USA, 2000.

[13] D. J. Levitin. Absolute memory for musical pitch: Evidence from the production of learned melodies. *Perception and Psychophysics*, 56(4):414–423, 1994.

[14] D. J. Levitin and P. R. Cook. Memory for musical tempo: Additional evidence that auditory memory is absolute. *Perception and Psychophysics*, 58(6):927–935, 1996.

[15] G. Li. Robust regression. In D. C. Hoaglin, F. Mosteller, and J. W. Tukey, editors, *Exploring Data Tables, Trends, and Shapes*, pages 281–343. Wiley and Sons, New York, 1985.

[16] A. T. Lindsay. Using contour as a mid-level representation of melody. Master's thesis, Mass. Institute of Technology, 1996.

[17] R. J. McNab, L. A. Smith, D. Bainbridge, and I. H. Witten. The New Zealand Digital Library MELody inDEX. *D-Lib Magazine*, May 1997. **http://www.dlib.org/dlib/may97/meldex/05witten.html**.

[18] R. J. McNab, L. A. Smith, I. H. Witten, C. L. Henderson, and S. J. Cunningham. Towards the digital music library: Tune retrieval from acoustic input. In *Proc. First ACM Conf. on Digital Libraries (DL'96)*, pages 11–18, Bethesda, MD, USA, Mar. 20–23 1996.

[19] M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

[20] S. Pauws. CubyHum: a fully operational query by humming system. In *Proc. 3rd Int'l Symposium on Music Information Retrieval (ISMIR'02)*, pages 187–196, Paris, 2002.

[21] E. D. Scheirer. Tempo and beat analysis of acoustic musical signals. *J. Acoust. Soc. Am.*, 103(1):588–601, Jan 1998.

[22] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[23] A. Uitdenbogerd and J. Zobel. Melodic matching techniques for large music databases. In *Proc. 7th ACM Conf. on Multimedia (MM'99)*, pages 57–66, Orlando, FL, USA, Oct. 30–Nov. 5 1999.

[24] Wildcat Canyon Software. **Autoscore Pro 2.0** computer software. Available (as of this writing) at **http://www.wildcat.com**.

[25] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.