Name That Tune: A Pilot Study in Finding a Melody From a Sung Query

Bryan Pardo and Jonah Shifrin
Department of Electrical Engineering and Computer Science, University of Michigan, 110 ATL, 1101 Beal Avenue, Ann Arbor, MI. 48109-2110. E-mail: bryanp&umich.edu

William Birmingham
Math & Computer Science Department, Grove City College—Faculty Box 2655, 100 Campus Drive, Grove City, PA 16127.

We have created a system for music search and retrieval. A user sings a theme from the desired piece of music. The sung theme (query) is converted into a sequence of pitch-intervals and rhythms. This sequence is compared to musical themes (targets) stored in a database. The top pieces are returned to the user in order of similarity to the sung theme. We describe, in detail, two different approaches to measuring similarity between database themes and the sung query. In the first, queries are compared to database themes using standard string-alignment algorithms. Here, similarity between target and query is determined by edit cost. In the second approach, pieces in the database are represented as hidden Markov models (HMMs). In this approach, the query is treated as an observation sequence and a target is judged similar to the query if its HMM has a high likelihood of generating the query. In this article we report our approach to the construction of a target database of themes, encoding, and transcription of user queries, and the results of preliminary experimentation with a set of sung queries. Our experiments show that while no approach is clearly superior to the other system, string matching has a slight advantage. Moreover, neither approach surpasses human performance.

Introduction

As digital music proliferates, researchers have been investigating methods to easily and accurately search musical databases. Query-by-humming (QBH) systems allow users to pose queries by singing or humming them. This approach has two advantages: first, humming or singing is a natural way to describe the melody of a piece of music; second, QBH systems search musical content. This second point is in stark contrast to approaches to music retrieval based on searching music metadata such as search by song title, genre, and so forth.

Our group has created a system for music search and retrieval called MuseArt (Birmingham et al., 2002; Shifrin et al., 2002). In this system, a user sings a query, assumed to be a theme, hook, or riff from the piece of music the user wants to find. The query is sung into audio recording software, transcribed, and then matched to a database of musical themes. The matcher returns a song list, ranked by similarity, which is brought up in Apple's iTunes audio playback software. The user may then click on the songs in the list to play the desired piece of music. Figure 1 gives an overview of the system's operation. In this article, we detail the matching processing.

Related Work

Music information retrieval aimed at returning the best match from a database for a query has been investigated by a number of researchers. A good source for finding articles on the topic is the music-ir Web site at www.music-ir.org. The dominant query matching techniques investigated in the literature have been n-grams and simple dynamic-programming-based string matching.

Pickens (2000) compared language models to n-grams by encoding subportions of database targets in the language model format and as various length n-grams and passing those to the search engine as queries. He found n-grams superior, however results from n-grams are only good when the full content of a piece in the database is passed in as a query. Downie and Nelson (2000) performed a systematic investigation of the best length n-gram to use, given queries and targets encoded as sequences of pitch intervals. Results were evaluated using normalized precision and recall measures. The best performance for simulated error-prone que-

Received January 10, 2003; revised August 13, 2003; accepted August 13, 2003

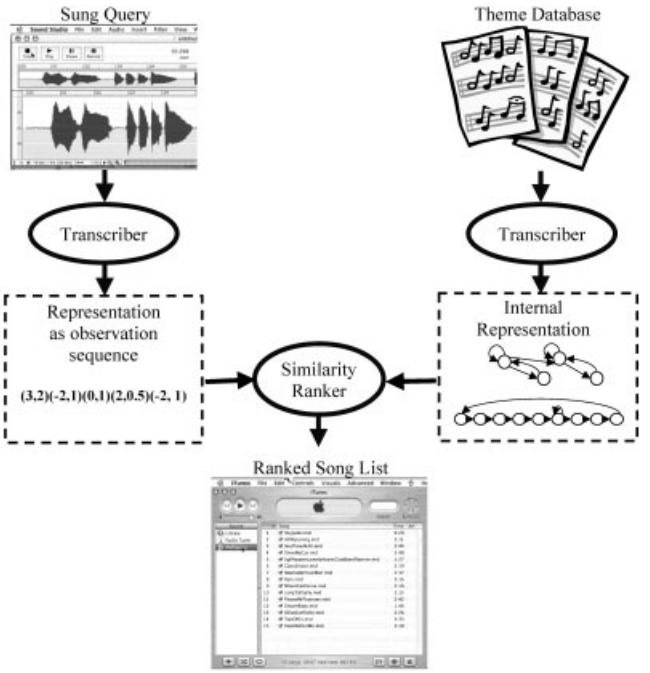

FIG. 1. System diagram.

ries was achieved using length four n-grams, as it provided the best fault tolerance for their simulated errors.

Uitenbogerd and Zobel (1999) examined the effect of encoding queries and targets using simple relative pitch contour, where intervals are categorized as ascending (U), descending (D), or the same (S), vs. encoding melodies as pitch intervals, or as modulo n pitch intervals. They also compare two kinds of n-gram measures to local string alignment with a fixed match score. Their results show pitch contour is the worst encoding. Modulo n and exact pitch intervals were both better and showed similar performance to each other. Local string alignment was found to beat the performance of n-gram based matching.

Query-by-humming music has been investigated by several research groups (Clarisse et al., 2002; Clausen et al., 2002; Doaisamy & Ruger, 2002; Hoos et al., 2001; McNab et al., 1996) in recent years, also with an emphasis on n-gram and string-matching techniques. McNab et al. (1996) describe approximate string-matching techniques and dynamic programming using a simple match score function. They analyze how many notes are required to uniquely identify melodies in their database, given different encoding schemes and exact vs. approximate matching. They do not present any results on system performance when presented with real sung queries, however.

A number of systems, such as those of Kornstadt (1998) and Tseng (1999), have focused on sound input, and thus are particularly relevant to our work. These systems generally take as input a melody that is "hummed" or played on some type of musical input device, such as a MIDI keyboard. The hummed melodies are converted to text strings, usually with a representation of intervallic distance or simply relative pitch contour.

Some researchers have investigated stochastic methods, particularly the use of Markov models for music retrieval, but do not use hidden Markov models (Rabiner & Juang, 1993), forcing the system to require exact matches between query and theme (Durey & Clements, 2001). This is a problem, since hummed input is usually fraught with a number of distortions inherent in the way that humans remember and perform melodies. Such distortions include a tendency to raise or lower the pitch of various notes, a tendency to "go flat" over time, losing the beat, or humming "off key." Unfortunately, these distortions plague even the finest musicians (Meek & Birmingham, 2002; Raphael, 1999).

To account for these distortions, the query must be treated as imprecise. The system described here assumes matching based solely on timing and pitch contour. The representation of pitch and timing is robust to differences in the tempo and transposition of the query, as compared with systems such as that developed by Durey (Durey & Clements, 2001), which relies on the target and query being presented at the same tempo and in the same pitch range.

We match queries to musical themes in one of two ways. The first is with standard string-matching techniques that have been adjusted to allow for errors in the query and probabilistic matching of elements between the query and target. The other method of matching we use is the hidden Markov model (HMM), which is designed to handle discrepancies (such as singer error) between expected and observed sequences. The query is treated as an observation sequence and a theme is judged similar to the query if the associated HMM has a high likelihood of generating the query. With both methods, pieces in the database are returned to the user in order, ranked by similarity. A piece of music is deemed a good match if at least one theme from that piece is similar to the query.

Representation of a Query

A query is a melodic fragment sung by a single individual. The singer is asked to select one syllable, such as "ta" or "la," and use it consistently during the query. The consistent use of a single consonant–vowel pairing lessens pitch-tracker error by providing a clear onset point for each note, as well as reducing error caused by vocalic variation.

A query is recorded as a .wav file and is transcribed into a MIDI-like representation using a pitch-tracking system written by our research group, based on an enhanced autocorrelation algorithm (Tolonen & Karjalainen, 2000).

MIDI is to a digital audio recording of music as ASCII is to a bitmap image of a page of text. Note events in MIDI are specified by three integer values in the range 0 to 127. The first value describes the event type (e.g., "note off" and "note on"). The next value specifies which key on a musical keyboard was depressed. Generally, middle "C" gets the number 60. The final integer specifies the velocity of a note (used as an indication of loudness).

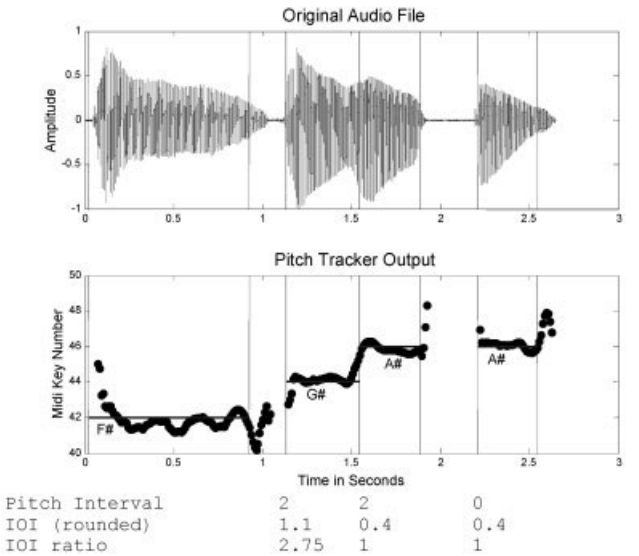

FIG. 2. A transcription of a sung query.

Pitch tracking can be thought of as the equivalent of character recognition in the text world. In our system, the pitch tracker divides the input file into 10 millisecond frames and tracks pitch on a frame-by-frame basis. Contiguous regions of at least five frames (50 millisecond) whose pitch varies less than one musical half step are called notes. The pitch of each note is the average of the pitches of the frames within the note. The pitch tracker returns a sequence of notes, each of which is defined by pitch, onset time, and duration. Pitches are quantized to the nearest musical half step and represented as MIDI pitch numbers. We define the following.

- $pitchInterval_n$ is the difference in pitch between note n and note $n + 1$.
- The *inter onset interval* (IOI_n) is the difference between the onset of notes n and $n + 1$.
- $IOIratio_n$ is IOI_n/IOI_{n+1}. For the final transition, $IOIratio = 1$.
- A note transition between note n and note $n + 1$ is described by the duple $\langle pitchInterval, IOIratio \rangle$.

Figure 2 shows a time-amplitude representation of a sung query, along with example pitch-tracker output and a sequence of values derived from the MIDI representation (the *pitchInterval*, *IOI*, and *IOIratio* values). Time values in the figure are rounded to the nearest 100 milliseconds.

We represent a query as a sequence of note transitions. Note transitions are useful because they are robust in the face of transposition and tempo changes, as we explain in the Appendix. Note that the note transition representation does not allow direct representation of rests, since it does not make sense to discuss the pitch interval between a note and a rest. Instead, rests simply increase the *IOI* between two note onsets.

String-Alignment Methods

A string is any sequence of characters drawn from an alphabet. String matchers find the best alignment between string A and string B by finding the lowest cost (or, equivalently, highest reward) transformation of A into B in terms of operations (matching or skipping characters). Dynamic-programming based implementations that search for a good alignment of two strings have been used for over 30 years to align gene sequences based on a common ancestor (Needleman & Wunsch, 1970), and have also been used in musical score following Danneberg (1984), Puckette and Lippe (1992), and Pardo and Birmingham (2001; 2002) and query matching (Hu et al., 2002).

The Global Alignment Algorithm

The Global alignment algorithm is a typical string alignment, or string-matching, algorithm. Denote the length of a string, S, as $|S|$. Let there be the query string, Q, and the target string, T. Assume they are both composed of characters drawn from the same alphabet. Construct a matrix *AlignScore* with $|Q| + 1$ rows and $|T| + 1$ columns where *AlignScore*(i,j) is the score of the best alignment between the initial segment q_1 through q_i of Q and the initial segment t_1 through t_j of T.

The process is initialized by setting *AlignScore*$(0,0) = 0$. Thereafter, the elements of the matrix are filled in using Equation 1. The top line in this equation gives the reward assigned in the score matrix for calling (q_i,t_j) a match. The middle line calculates the penalty for skipping target element t_j and the lowest line finds the penalty for skipping query element q_i. Note, when in column 0 or row 0 one of the lines in the equation is undefined. We treat undefined values as negative infinity.

$$AlignScore(i,j) = \max \begin{cases} AlignScore(i-1, j-1) + matchScore(q_i, t_j) \\ AlignScore(i-1, j) - skipPenaltyTarget(t_j) \\ AlignScore(i, j-1) - skipPenaltyQuery(q_i) \end{cases} \quad (1)$$

We define a simple example *matchScore* function as follows:

$$matchScore(q_i, t_j) = \begin{cases} 2, & \text{if } q_i = t_j \\ -2, & \text{otherwise} \end{cases} \quad (2)$$

Let the penalty for skipping an element of either sequence be given by:

$$\begin{aligned} skipPenaltyQuery(q_i) &= 1 \\ skipPenaltyTarget(t_j) &= 1 \end{aligned} \quad (3)$$

As *AlignScore* is filled in, another table may be kept, keeping track of the parent used to fill in each cell (i,j). In many cases, it may be that more than one of the parents of cell (i,j) gives the maximum value. In this case, all parents may be noted (although, in practice, researchers have noted

Target / Query		G	A	B	B
	0	-1	-2	-3	-4
G	-1	**2**	1	0	-1
D	-2	**1**	0	-1	-2
A	-3	0	**3**	**2**	-1
C	-4	-1	**2**	**1**	0
B	-5	-2	1	**4**	**3**

FIG. 3. Alignment matrix.

only a single parent). Once this is done, the best-scoring alignment(s) may be read by starting at the final cell in the matrix and tracing backwards through the series of parents used to generate the score.

An *AlignScore* matrix for query "G D A C B" and target "G A B B" is shown in Figure 3. Values for the matrix were calculated using Equations 1–3. Target elements identify columns. Query elements identify rows. Arrows show the parent(s) of each element along a maximal-scoring alignment. Where an element has multiple parents that maximize Equation 1, arrows to all parents are shown.

In the figure, a vertical arrow indicates a skip of an element in the query sequence, a horizontal arrow is a skip of a target element, and a diagonal arrow indicates a match between the two. Cells along the maximal-scoring alignment path(s) are shown in boldface with arrows marking the path. The score for the top global alignment is given by the value in the lower, right-hand corner of the table. In this case, the score is 3. The table shows the following four maximal-scoring alignments between query and target. Here, matches are aligned vertically and a dash indicates a skip.

```
TARGET:  G-A-BB  G-A-BB  G-ABB  G-AB-B
QUERY:   GDACB-  GDAC-B  GDACB  GDA-CB
```

Modeling Transcription and Singing Error

The string matcher described previously has a simplistic *matchScore* function, giving two points if q_i is an exact match of t_j and subtracting two if they are at all different. This is too simple a model when comparing a note sequence transcribed from audio using a pitch tracker to a target note sequence (a theme). Octave displacement, tracking a strong partial, and half step errors due to pitch quantization are all common occurrences (Sterian, 1999). Singers are also prone to systematic error that can be predicted. Such errors can be handled gracefully if a probability distribution over the set of possible observations (the note reported by a pitch tracker), given a state (the note in the theme), is maintained. It would be good to have a *matchScore* function that could take this kind of error into account.

The dynamic-programming algorithm introduced by Gotoh (1982) as described in Durbin et al. (1998) finds an optimal global alignment between two strings, Q and T. It takes into account the likelihood that any given element in sequence Q is related to any given element in sequence T. We adapt Gotoh's approach to the problem of modeling transcription error and replace the *matchScore* function with the log-odds ratio *matchScore* given by Equation 4.

$$matchScore(q_i, t_j) = \log\left(\frac{P(q_i, t_j|match)}{P(q_i, t_j|random)}\right) \quad (4)$$

This function returns a negative value when the probability of a meaningful match is below that of a random co-occurrence. Similarly, the value is positive when a meaningful match is more likely than random chance.

Let there be two distinct, but identical alphabets, A_{query} and A_{target}. They might contain, for example, the 88 pitches available on a piano, or a fixed set of ⟨*pitchInterval, IOIratio*⟩ duples.

Assume every element, e, of each alphabet occurs independently with some known prior probability, $P(e)$. For A_{target}, $P(e_{target})$ may be estimated by the proportion of times e_{target} occurs in a representative corpus of targets. This is useful domain knowledge. For example, music for a baritone singer will not contain many pitches three octaves above middle C. For $A_{query,}$ $P(e_{query})$ for each e_{query} may be estimated from a representative corpus of transcriptions.

$$P(e_{target}, e_{query}|random) = P(e_{target})P(e_{query}) \quad (5)$$

The prior probability of a random co-occurrence of e_{target} and e_{query} is given by Equation 5. Typically, $P(e_{target}, e_{query}|$ *random*) is calculated using the naïve assumption that all members of A_{target} have an equal probability of occurrence, as do all members of A_{query}. This results in a fixed probability for all pairings of e_{target} and e_{query} as follows.

$$\forall\, (e_{target} \in A_{target}, e_{query} \in A_{query}), P(e_{target}, e_{query}|random) = \frac{1}{|A_{target}| \cdot |A_{query}|} \quad (6)$$

The probability of a match is given by Equation 7. Here, one must estimate the likelihood of the transcriber reporting e_{query} when the singer intends e_{target}. This probability captures the combined error introduced by the singer and the transcription device. We describe our method of estimating this error here in a later section, entitled "Making Hidden Markov Models." For an example of how this was done for alto saxophone performance, see Pardo and Birmingham (2002).

$$P(e_{target}, e_{query}|match) = P(e_{query}|e_{target}) \quad (7)$$

Local String Alignment

The Global algorithm generates a score for the best global alignment between two sequences. This is an appropriate approach if one assumes that the likely match scenario is finding two complete copies of the same sequence, so every element of both sequences must be explained in terms of a match or a skip. Of course, we cannot be assured that a person searching for a song will sing the exact theme in the database. It is more likely that there will be some significant overlap between the query and theme, but that each will have a beginning or end portion not covered by the other. In this case, it is more appropriate to use a local string-matching algorithm. We chose to use the one given described in Durbin et al. (1998).

$$AlignScore(i, j) = \max \begin{cases} 0 \\ AlignScore(i-1, j-1) + matchScore(q_i, t_j) \\ AlignScore(i-1, j) - skipPenaltyTarget(t_j) \\ AlignScore(i, j-1) - skipPenaltyQuery(q_i) \end{cases} \quad (8)$$

The Local alignment algorithm replaces Equation 1 with Equation 8. Once the *AlignScore* matrix is full, the score of the best alignment of a subsequence of Q to a subsequence of T is used to judge the similarity of Q and T. This is done by returning the largest value in the *AlignScore* matrix, wherever it occurs.

Selecting the Most Likely Target

Let there be a query, Q, and a set of targets, $\{T_1 \ldots T_n\}$. An order may be imposed on the set of targets by running the same alignment algorithm (Global or Local) between Q and each target, T_i, and then ordering the set by the value returned, placing higher values before lower. We take this rank order to be a direct measure of the relative similarity between a theme and a query. The *i*th target in the ordered set is then the *i*th most like the query. Thus, the first target is the one most similar to the query.

The computational complexity of finding the most similar target to the query depends on the time it takes to compute the similarity of a single target. In the case of string alignment, the number of steps required to find the similarity of a target to the query is linearly related to the size of the alignment table. This, of course, is just the length of the target multiplied by the length of the query. If we let N be the number of targets in the database, and $mean(|T|)$ be the average length of a target, then the computational complexity of finding the most similar target is given by Equation 9.

$$StringMatchComplexity = O\left(\sum_{i=1}^{N} |Q| \cdot |T_i|\right) = O(N|Q|mean(|T|)) \quad (9)$$

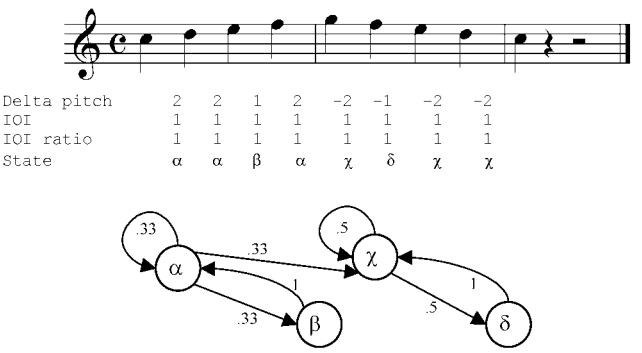

FIG. 4. Markov model for a scalar passage.

Targets as Markov Models

A Markov model (MM) models a process that goes through a sequence of discrete states, such as notes in a melody. The model is a weighted automaton that consists of:

- A set of states, $S = \{s_1, s_2, s_3, \ldots, s_n\}$.
- A set of transition probabilities, T, where each $t_{i,j}$ in T represents the probability of a transition from s_i to s_j.
- A probability distribution, π, where π_i is the probability the automaton will begin in state s_i.
- E, a subset of S containing the legal ending states.

In this model, the probability of transitioning from a given state to another state is assumed to depend only on the current state. This is known as the Markov property.

The directed graph in Figure 4 represents a Markov model of a scalar passage of music. States are note transition duples, ⟨*pitchInterval, IOIratio*⟩. Nodes represent states. Recall that the note transition representation does not allow direct representation of rests, as all states describe pitch and rhythm intervals between note onsets.

We assume all states are equally likely to be the start state. As a default, we currently assume all states are legal ending states. Directed edges represent transitions. Numerical values by edges indicate transition probabilities. Only transitions with non-zero probabilities are shown.

Here, we have implicitly assumed that whenever state s is reached, it is directly observable, with no chance for error. This is often not a realistic assumption. There are multiple possible sources of error in generating a query. The singer may have incorrect recall of the melody he or she is attempting to sing. There may be production errors (e.g., cracked notes, poor pitch control). The transcription system may introduce pitch errors, such as octave displacement, or timing errors due to the quantization of time. Such errors can be handled gracefully if a probability distribution over the set of possible observations (such as note transitions in a query) given a state (the intended note transition of the singer) is maintained.

A model that explicitly maintains a probability distribution over the set of possible observations for each state is

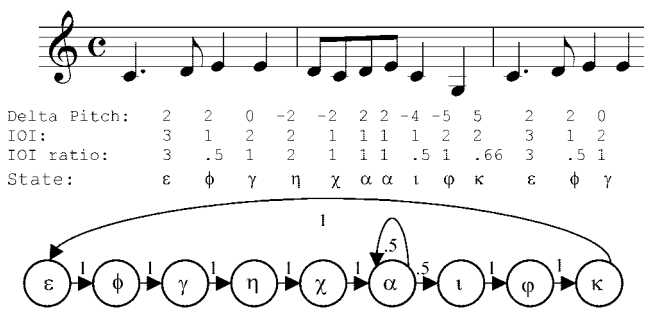

FIG. 5. Markov model for *Alouette* fragment.

called a hidden Markov model (HMM). More formally, an HMM requires two things in addition to that required for a standard Markov model:

- A set (alphabet) of possible observations, $A_{observe} = \{o_1, o_2, o_3, \ldots, o_n\}$.
- A probability distribution over the set of observations for each state in *S*.

In our approach, a query is a sequence of observations and $A_{observe}$ corresponds directly to A_{query} for the string matchers. Each observation is a note-transition duple, ⟨*pitchInterval, IOIratio*⟩. Musical themes are represented as hidden Markov models.

Making Markov Models From MIDI

Our system represents musical themes in a database as HMMs. Each HMM is built automatically from a MIDI file encoding the theme. The unique duples characterizing the note transitions found in the MIDI file form the states in the model. Figure 4 shows a passage with eight note transitions characterized by four unique duples. Each unique duple is represented as a state.

Once the states are determined for the model, transition probabilities between states are computed by calculating what proportion of the time state *a* follows state *b* in the theme. Often, this results in a large number of deterministic transitions.

An example of this is shown in Figure 5, where only a single state has two possible transitions, one back to itself and the other on to the next state.

Markov models can be thought of as generative models. A generative model describes an underlying structure able to generate the sequence of observed events, called an observation sequence.

Note that there is not a one-to-one correspondence between model and observation sequence. A single model may create a variety of observation sequences, and an observation sequence may be generated by more than one model. Recall that our approach defines an observation as a duple, ⟨*pitchInterval, IOIratio*⟩. Given this, the observation sequence $q = \{(2,1), (2,1), (2,1)\}$ may be generated by the HMM in Figure 4 or the HMM in Figure 5.

Researchers (Durey & Clements, 2001; Hoos et al., 2001) have used Markov models for query matching that were generated in a manner similar to that described above. Such systems are vulnerable to query error, for if a single transition is present in the query that is not present in the model the model reports a zero probability of having generated the query. Hoos et al. (2001) have attempted to deal with this by introducing low-probability transitions into a Markov model where no note-transitions were observed in the theme upon which the model was based. This allows inexact matches between models and queries to return non-zero probabilities. There is no correlation, however, between the strength of a low-probability transition and the likelihood of a particular error. This makes the approach a blunt instrument that does not take into account specific errors (such as a singer that regularly confuses perfect fifths with perfect fourths) that are known to occur. Such errors are handled well by a hidden Markov model.

Making Hidden Markov Models

To make use of the strengths of a hidden Markov model, it is important to model the probability of each observation o_i in the observation alphabet, $A_{observe}$, given a hidden state, *s*. The approach described here is also used to estimate values for the function in Equation 7.

In our system, observations consist of duples ⟨*pitchInterval, IOIratio*⟩. There are 12 musical half steps in an octave. If one assumes pitch quantized at the half step and that a singer will jump by no more than an octave between notes, there are 25 possible *pitchInterval* values, corresponding to all possible transitions whose range is no greater than an octave. We quantize *IOIratio* to one of five values. We chose five values, as an earlier study indicated that roughly five bins were the minimum number that could be used without compromising performance. This means there are $25 \cdot 5 = 125$ possible observations, given a hidden state. Since hidden states are also characterized by ⟨*pitchInterval, IOIratio*⟩, there are 125 possible hidden states for which observation probabilities need to be determined. The resulting table has 125^2, or over 15,000 entries.

We use a typical method to estimate probabilities. A number of observation, hidden-state pairs are collected and observed frequencies are used as an estimator for expected probabilities. Given a state, *s*, the probability of observation o_i may be estimated by the count of how often o_i is seen in state *s*, compared to the total number of times *s* is encountered.

$$P(o_i|s) = \frac{count(o_i, s)}{\sum_{j=1}^{|O|} count(o_j, s)} \tag{10}$$

Creating a data set of paired observations and hidden states from which to estimate over 15,000 probabilities is daunting. This can be made more tractable by assuming conditional independence between *pitchInterval* and *IOIra-*

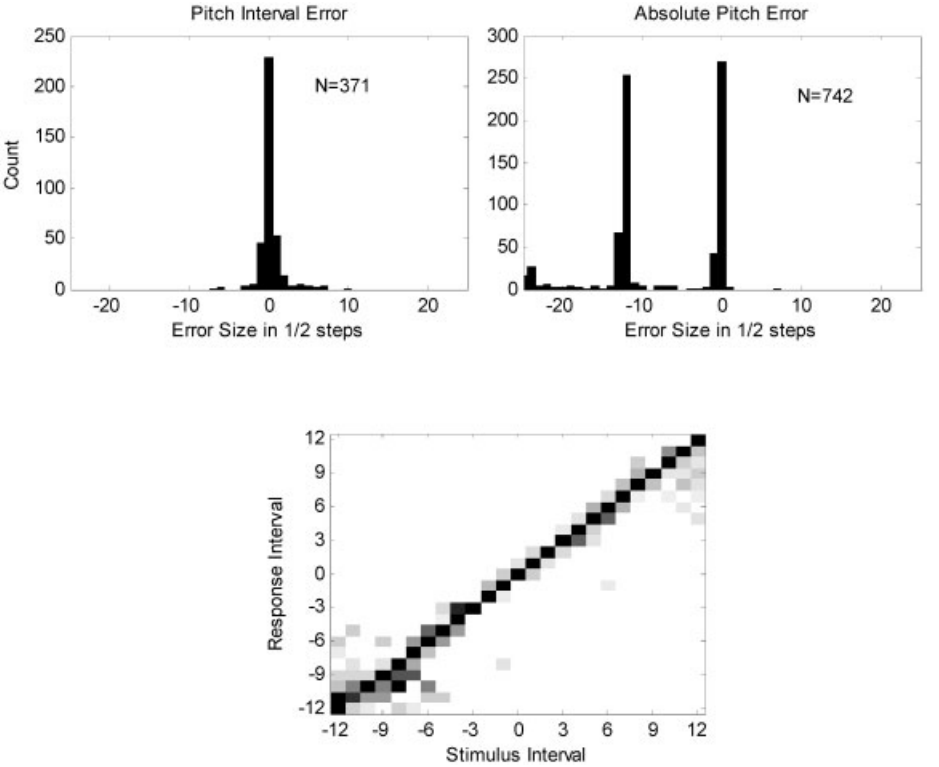

FIG. 6. Error models for a singer.

tio. Were we to represent rhythm as *IOI*, this might not be a reasonable assumption, since it is easier to quickly sing a small pitch interval (such as a semi-tone) than it is to quickly sing a large one (such as 10 semi-tones). We, however, represent the rhythm as *IOIratios*. There is no obvious correlation to be drawn between the size of the current pitch interval and the relative duration of this pitch interval when compared to the previous pitch interval's duration.

Given independence, two separate observation probability tables may be created, one for *pitchInterval* and one for *IOIratio*. The former having $25 \cdot 25 = 625$ values, the latter having $5 \cdot 5 = 25$ values. The probability of encountering any observation duple, given a hidden-state duple, can then be derived from the two tables using Equation 11.

$$P(o|s) = P(pitchInterval_o|pitchInterval_s) \times P(IOIratio_o|IOIratio_s) \quad (11)$$

Even given the reduced size of the observation probability tables allowed by Equation 4, there are observations that do not occur in the training data. We cannot assume that a pairing unobserved in the training set will never be observed in an actual query. Thus, we impose a minimal probability, P_{min} on both observation probability tables. Any probability falling below P_{min} is set to P_{min}. Probabilities are then normalized so that the sum of all observation probabilities, given a particular hidden state, is equal to one.

Estimating pitch interval observation probabilities.

Our system uses labels for hidden state *pitchInterval* values that are integers in the range from negative 12 to positive 12. This corresponds to all musical intervals from a descending octave through an ascending octave. Our current method of deriving pitch interval observation probability for a particular singer is to present the singer with hundreds of examples of ascending and descending intervals of one octave or less in the range from the second E below middle C (MIDI key 40) to the second G above middle C (MIDI key 79). This range was chosen to approximate the range of pitches commonly encountered in vocal music, from baritone through soprano voices.

After hearing an interval, the person sings that interval into a microphone and the result is automatically pitch-tracked and segmented. The output interval is then compared to the input interval and the resulting pairing used to update the observation probability table. We do not attempt to estimate the singer's vocal range before training and do not limit the intervals presented to the range the singer is capable of reproducing. This is because there will be many instances when a singer will attempt to reproduce something outside of his or her range and it is important to model what the singer does when this happens.

Figure 6 shows three views of combined singer-pitch tracker error on the training set for a particular singer. The singer in this example is a 22-year-old male, computer science major with no vocal training. The upper left figure shows a histogram of pitch-interval error as measured in half-steps. In the pitch-interval histogram, only the size of

the pitch interval contributes to error. Here, if the system presents the interval 42 to 44 and the person sings 56 to 58, it is counted as an error of size 0 (i.e., not an error). Similarly, the absolute-pitch error histogram shows the absolute-pitch error. Here, the person singing 56 to 58 would generate two errors, each of size 14.

The graph in the bottom portion of the figure is the observation probability matrix for the singer. It is represented as a confusion matrix. The horizontal axis represents the stimulus pitch interval presented to the singer. The vertical axis represents the response generated by the combination of singer and pitch-tracker. Each square represents the frequency with which a particular stimulus–response pair was observed. The darker the square, the more frequent the occurrence. Response intervals > 12 are binned to 12. Those < -12 are binned to -12.

The confusion matrix in shows the singer's tendency to become increasingly error prone as the size of the stimulus interval increases. This is captured in our observation probability model.

Estimating IOIratio observation probabilities. Research in music perception indicates rhythmic ratio values fall naturally into evenly spaced bins in the log domain. Given this, we decided to measure error in the rhythm domain as the log of a ratio of *IOIratio* values as shown in Equation 12.

$$error = \ln\left(\frac{\text{expected IOIratio}}{\text{observed IOIratio}}\right) \tag{12}$$

Here, if the expected *IOIratio* is equal to the observed *IOIratio*, error is zero. When observed *IOIratio* is less than expected, the error is positive, when observed error is more than expected, the error is negative. Error was initially quantized to 27 values, spaced evenly on a logarithmic scale. The number of bins was chosen ad hoc and was an artifact of selecting a bin size of 0.2. We then determined the number of bins could be reduced to five without degrading the ability of a matcher to find the right target for a musical query.

Figure 7 shows observed *IOIratio* error for a typical singer using our initial 27 bins. Here, the height of the bar represents the relative frequency of occurrence of an error of the given size in the training data. We performed a pilot study of *IOIratio* error and found the error distribution, when measured using Equation 6, appeared not to vary greatly with the expected *IOIratio*. For example, this singer's error distribution approximates a Gaussian function that has a median of zero and a standard deviation of 0.28, regardless of expected *IOIratio*. Given this assumption, we use the same distribution for all expected *IOIratios*.

Using the observation probabilities with a Markov model. Given probability tables for *IOIratio* and *pitchInterval*, a Markov model constructed from a MIDI file, such as the one in Figure 5, may be treated as a hidden Markov model by relating the states of the model to the observations through the observation probabilities.

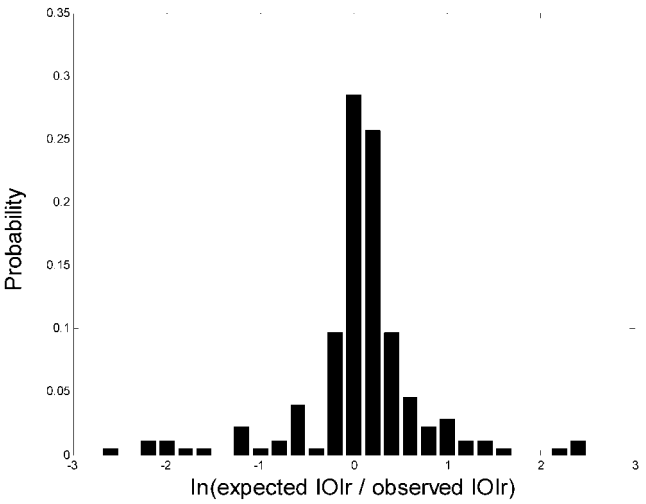

FIG. 7. Error model for *IOIratio*.

A query is an observation sequence and the probability of an observation, given a state, may be calculated using the observation probability table. Consider the first observation in the query transcription from Figure 2 and the Markov model in Figure 5. What is the probability that the observation (*pitchinterval* = 2, *IOIratio* = 2.75) was generated by the first (leftmost) state in the model, given the observation probability tables for our example singer?

The first state in the model represents the duple (*pitchinterval* = 2, *IOIratio* = 3). The *pitchinterval* observation-probability table reports a $P(\text{observed} = 2 \mid \text{expected} = 2) = 0.76$.

The log of the expected *IOIratio* divided by the observed *IOIratio* = $\ln(3/2.75) = 0.0870$. This value falls into the bin for 0 in the error model from Figure 7. The probability of falling in this bin is 0.29.

Multiplying these probabilities together in accordance with Equation 4, we get the probability of the observation, given the hidden state $0.76 \cdot 0.29 = 0.22$.

This process can be repeated for each combination of observation and hidden state in a model. The likelihood any particular path through the model generated the observation sequence may then be determined by multiplying the observation probabilities by the transition probabilities from state to state.

The Forward Algorithm

The Forward algorithm (Rabiner & Yuang 1993), given an HMM and an observation sequence, returns a value between 0 and 1, indicating the probability the HMM generated the observation sequence. Given a query length, $|Q|$, the algorithm takes all paths through the model of up to $|Q|$ steps. The probability each path has of generating the observation sequence is calculated and the sum of these probabilities gives the probability that the model generated the

observation sequence. This algorithm takes on the order of $|S|^2|Q|$ steps to compute the probability, where $|S|$ is the number of states in the model. In our system, the number of states is equal to $|T^u|$, the number of unique elements in the target string, T. The value of $|T^u|$ is limited to a maximum of the number of elements in A_{target}, the target alphabet.

No Baum-Welch Training

Those familiar with the HMM literature may wonder why we do not use a standard parameter re-estimation technique such as Baum-Welch training. Our method of generating hidden Markov Models requires only a single exemplar, paired with a known error model. Typical HMM training requires a number of examples. Building a large database of target songs with many sung examples of each song is extremely time consuming and is a great hindrance on the scalability of a song-finding system. One could, of course, make artificial examples using a known error model and then train the HMM in the standard fashion. This would, however, be an inefficient approach in that it would be approximating a known error model. Thus, we chose to directly use the error model.

Finding the Best Target

We encode the themes in our database as HMMs and the query is treated as an observation sequence. Given this, we are interested in finding the HMM most likely to generate the observation sequence. This can be done using the Forward algorithm.

Let there be an observation sequence (query), O, and a set of models (themes), M. An order may be imposed on $\mathbf{M}$ by performing the Forward algorithm on each model m in M and then ordering the set by the value returned, placing higher values before lower. The ith model in the ordered set is then the ith most likely to have generated the observation sequence. We then take this rank order to be a direct measure of the relative similarity between a theme and a query; the first theme is the one most similar to the query.

While this is simple enough in principle, in practice there are constraints on how one may compare HMMs for the purpose of finding the best target in the database. First, the above method makes the implicit assumption all targets have an equal likelihood of occurrence. If it is known that 99% of all queries to a database of Beatles music will be for the song, "Michelle," it would make sense to skew the results of the ranking by a weighting factor based on that fact. Currently, we do not have an adequate model for the likelihood of any given target. So, instead we use the naïve assumption of equal probability for all targets.

Another problem is comparing "apples to oranges." If two models have a different mapping from hidden state to observation, it may not make sense to say that one is more likely than the other to have generated the observation sequence. For example, imagine one HMM that maps from note observation sequences onto hidden states representing rhythm, while another maps onto pitch. It may not make sense, depending on the task, to directly compare the probabilities generated by these two HMMs. This is not an issue for our system, because all models in our database use the same observation probabilities when they are compared to each other.

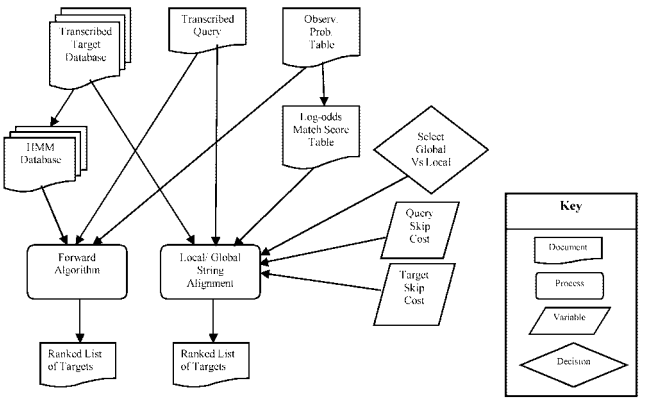

FIG. 8. Experimental setup.

As with the string matcher, the computational complexity of finding the most similar target to the query depends on the time it takes to compute the similarity of a single target.

$$HMMmatchComplexity = O\left(\sum_{i=1}^{N} |Q| \cdot |T_i^u|^2\right) = O(N|Q|mean(|T^u|^2)) \quad (13)$$

Experimental Set-Up

We have presented two string-matching methods and one HMM method for ranking similarity between targets (musical themes) in a database and a sung query. This begs the question "Which method is better?" To answer this question, we assembled a database of Beatles songs, a small set of test queries and then ran the methods head-to-head, to determine which one was most likely to return the right answer as the top pick. Both the query set and the database size are limited, so these results are provisional; however, they do indicate clear trends that bear reporting, especially with respect to relative efficacy of error models.

Figure 8 illustrates the data flow for the experimental setup for comparing the HMM-based Forward algorithm to the Global and Local string-alignment algorithms. For a particular query, the query, target database, observation probability table (and thus, match score matrix), and skip costs are all fixed. The targets are then ranked by the Forward, Global, and Local algorithms. The three algorithms may then be compared by how well the correct target ranked in their output. A single trial ranks all targets in the database against all queries in the query set, given a fixed observation probability model (match score matrix), fixed skip costs, and ranking algorithm (Global, Local, or Forward). We now describe the experimental setup in greater detail.

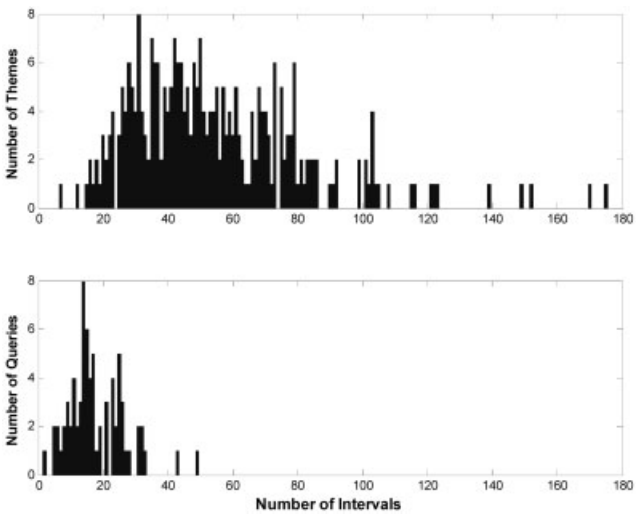

FIG. 9. Histogram of theme and query lengths.

Transcribed Target Database

We collected a corpus of 260 pieces of music encoded as MIDI from public domain sites on the Internet. The corpus is composed entirely of pieces that have been recorded by the Beatles. This includes all pieces recorded on albums for U.S. release and a number of "covers" they performed that were originally composed by other artists, such as "Roll Over Beethoven." For a full list of pieces in the database, please consult the MusEn Web site at http://musen.engin.umich.edu/.

We selected music performed by the Beatles because their music tends to be well known, the salient information to identify pieces tends to be melodic and relatively easy to sing, and the pieces are readily accessible, both as audio and as MIDI. Each piece in the corpus was represented in a database by a set of "themes," or representative monophonic melodic fragments. The number of distinct "catchy hooks," as determined by the third author, decided the number of themes chosen to represent each piece. Of the pieces, 238 were represented by a single theme, 20 by two themes, and two pieces were represented by three themes, resulting in a database of 284 monophonic themes. These themes constitute the set of targets in the database.

A sequence of ⟨*pitchInterval, IOIratio*⟩ pairs was created and stored in the database for each MIDI theme. Themes were quantized to 25 pitch intervals and five log-spaced *IOIratio* intervals. Each theme was indexed by the piece it was derived from. An HMM for the theme was then generated automatically from the theme sequence and placed in the database, using the method described herein.

The upper plot in Figure 9 contains a histogram of theme lengths. Here the horizontal dimension indicates theme length, and the vertical dimension indicates the count of themes of a given length.

Our database of Beatles themes has a mean number of 54.6 elements per sequence, with a mean of 20.8 unique elements per sequence. Comparing Equation 9 to Equation 13, it can be seen that the string matcher should perform faster than the HMM when the mean target length is less than $|T''|^2$, the square of the mean number of unique elements in a target. The mean value of $|T''|^2$ is 472.7 for our theme database. From this, one would expect an HMM to perform between 8 and 9 times slower than the equivalent string matcher. In our implementation, additional overhead in the HMM caused it to run roughly 12 times slower than the string matchers on the data set, with the string matchers taking an average of 0.16 seconds to process a single query and the HMM taking 1.97 seconds per query.

This leads us to the reason we chose to match queries to themes instead of full pieces. The first of these is speed. For the string matchers, the time required to process a given query is linearly related to the mean length of the targets in the database. The median theme length in the database is 49.5 intervals. This compares to a median of 2,701 intervals for the original MIDI files the themes were drawn from. Thus, the typical theme is over 50 times shorter than the typical piece and processing is 50 times faster. Second, the global string matcher is designed to work well when comparing sequences whose entire length is expected to be relevant to the comparison. This led us to believe matching performance would be improved by comparing only the most salient portions of the piece. Finally, our HMMs are constructed in a way that it is advantageous for them to be drawn from only the most salient portions of the piece. The longer a segment the HMM is drawn from, the greater the chance for spurious loops and branch points in the architecture, causing the model to give high matching scores to sequences not very similar to the sequence from which the model was derived.

Of course, the downside to the use of a limited number of short themes to represent a relatively lengthy work is database coverage. If a query is presented to the system and the query covers some portion of the piece not captured in its representative theme set, the system will obviously fail to retrieve the right piece. This is the price paid for the specificity and compactness of the themes.

Query Corpus

A query is a monophonic melody sung by a single person. Singers were asked to select one syllable, such as "ta" or "la," and use it consistently for the duration of a single query. The consistent use of a single consonant–vowel pairing was intended to minimize pitch-tracker error by providing a clear starting point for each note, as well as reducing error caused by dipthongs and vocalic variation.

Three male singers generated queries for the experiment. Singer 1 was a 22-year-old male with no musical training beyond private instrument lessons as a child. Singer 2 was a 27-year-old male with a graduate degree in cello performance. Singer 3 was a 35-year-old male with a graduate degree in saxophone performance. None were trained vocalists. All are North American native speakers of English.

TABLE 1. Observation probability model representational ability.

Index	Pitch interval model	*IOIratio* model	P_{min} value	Mean RR forward	Mean RR global
1	Singer 1	Singer 1	0	1	1
2	Singer 1	Singer 1	10^−5	1	1
3	Singer 1	Singer 1	10^−4	1	1
4	Singer 1	Singer 1	10∧−3	1	1
5	Singer 1	Singer 1	10^−2	1	1
6	Singer 1	Singer 1	10^−1	1.92	1
7	Singer 1	Even probability	0	1	1
8	Singer 1	Even probability	10^−5	1	1
9	Singer 1	Even probability	10^−4	1	1
10	Singer 1	Even probability	10^−3	1	1
11	Singer 1	Even probability	10^−2	1	1
12	Singer 1	Even probability	10^−1	3.14	1
13	Singer 3	Singer 1	0	1	1
14	Singer 3	Singer 1	10^−5	1	1
15	Singer 3	Singer 1	10^−4	1	1
16	Singer 3	Singer 1	10^−3	1	1
17	Singer 3	Singer 1	10^−2	1	1
18	Singer 3	Singer 1	10^−1	1.89	1
19	Singer 3	Even probability	0	1	1
20	Singer 3	Even probability	10^−5	1	1
21	Singer 3	Even probability	10^−4	1	1
22	Singer 3	Even probability	10^−3	1	1
23	Singer 3	Even probability	10^−2	1	1
24	Singer 3	Even probability	10^−1	2.82	1
25	Exponential decay	Exponential decay	NA	1	1
26	Exponential decay	Even probability	NA	1.02	1
27	Even probability	Exponential decay	NA	3.21	1
28	Perfect	Perfect	0	1	1
29	Unified exponential	Unified exponential	NA	1	1

Sung queries were recorded in 8 bit, 22.5 kHz mono using an Audio-Technica AT822 microphone from a distance of roughly 6 inches. Recordings were made directly to an IBM ThinkPad T21 laptop using its built-in audio recording hardware and were stored as uncompressed PCM.wav files.

Each singer was allowed a trial recording to get a feel for the process, where the recorded melody was played back to the singer. This trial was not used in the experimental data. Subsequent recordings were not played back to the singer. Once the trial recording was finished, each singer was presented with a list containing the title of each of the 260 Beatles recordings in our database. Each singer was then asked to sing a portion of every song on the list that he felt confident of being able to sing. Singer 1 sang 28 songs; Singer 2 sang 17; and, Singer 3 sang 28 songs. The result was a corpus of 73 queries covering 41 of the Beatles' songs, or roughly one sixth of the songs in our database. These queries were then automatically pitch tracked, segmented, and quantized to 25 pitch intervals and five *IOIratio* intervals with the same pitch tracking, segmentation, and quantization software used in estimating observation probabilities. This resulted in 73 observation sequences, composed of ⟨*pitchInterval*, *IOIratio*⟩ duples. These were used as the query set for all experiments. The lower plot in Figure 9 contains a histogram of the queries, sorted by query length. Mean query length was 17.8 intervals. The median length was 16. The longest query had 49 intervals, and the shortest had only two. The median number of unique elements per query sequence was nine.

Observation Probabilities and Match Scores

To observe the effects of different observation probability models, we used 29 different combinations of pitch interval and *IOIratio* observation probability models in our experiments. Six of these were variations on the observed pitch interval and *IOIratio* models of Singer 1 (see Figs. 6 and 7), with varying sizes of P_{min} pseudocounts for unobserved data. Six used the observed pitch interval values for Singer 1 along with an even probability distribution for *IOIratio*s. These, in effect, ignored rhythmic content in queries. Twelve observation probability tables were also constructed using pitch interval observations derived from a study of Singer 3. Five observation probability models were derived from synthetic distributions. Table 1 shows the combinations of observation probability models used.

The perfect distribution placed a one along the diagonal of the observation probability matrix and a zero in all other elements of the matrix. The exponential decay distribution for both pitch and rhythm models were derived by setting the values on the main diagonal of the observation probability matrix at one, elements one off the diagonal at 2^{-1}, elements two off the diagonal at 2^{-2}, elements at three off

the diagonal at 2^{-3}, and so on. Values were then renormalized by row to sum to one. The unified exponential distribution was derived by creating a single, unified matrix of 125×125 elements and creating an exponential decay from the diagonal.

Once the observation probability matrices were created, a log-odds match score matrix was created from each of the observation probability matrices in Table 1. This was done simply by making the naïve assumption from Equation 6 and then applying Equation 4. This allows for direct comparison between a ranker using HMMs with a given observation probability table and a string matcher using the match score matrix derived from that observation probability table.

Evaluation Method

We have 284 themes in the database representing 260 songs. The rank of a song is the rank of its best scoring theme. The right rank of a query is the rank of the correct song for the query. The mean right rank for a trial is the average right rank for all queries in the trial. While this is often a good measure of performance, it can be sensitive to poorly ranking outliers. We can capture the same information in a manner that is slightly less sensitive to this by using mean reciprocal right rank (MRRR), which is calculated in Equation 14.

$$MRRR = \frac{\sum_{n=1}^{numberOfQueries} \frac{1}{rightRank_n}}{numberofQueries} \quad (14)$$

Mean reciprocal right rank is the inverse of the mean right rank. For MRR, higher values are better and they range between 1 and 1/260 (the number of songs in the database).

Experimental Results

Test 1: Observation Probability Model Representational Ability

As a test of the representational ability of the 29 observation probability models, we first ran our trials using the targets in the database as the query set. Thus, there were no problems of database coverage or incomplete or imperfect matches between the ideal target and the query. In this case, an effective ranker, with distinctive targets and a good representation, should return a mean right rank of one.

We ran trials on all 29 tables using the Forward algorithm. We then used the log-odds match score matrices generated from these tables, fixed both the target and the query skip cost at five and then ran trials on all 29 tables using the Global string-alignment algorithm (since queries match targets exactly in this trial, there is no difference between the use of Global or Local string alignment).

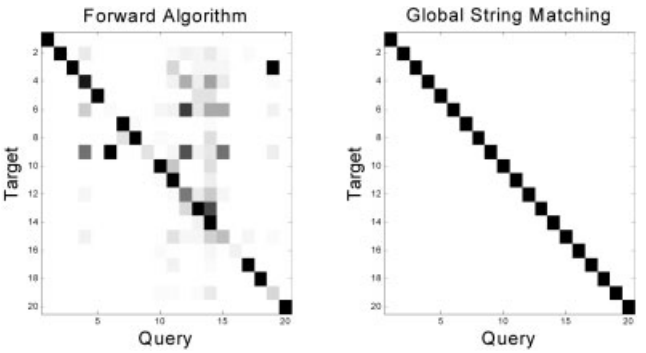

FIG. 10. Confusion matrices for Test 1, observation probability model 27.

The mean right rank (MRR) for this test is shown in Table 1, broken down by observation-probability model and matcher. Using the string-alignment algorithm, the MRR was always one. Using the Forward algorithm, 23 out of 29 of the representations generated a mean right rank of one. The worst case was that of observation-probability model 27, which uses an exponentially decaying probability for the rhythm representation and an even probability distribution for pitch. This is equivalent to ignoring pitch contour.

Figure 10 shows the confusion matrix for the first 20 themes in the target database for both the Forward algorithm and the Global alignment algorithm, when using observation probability model 27. Here, each column corresponds to a query theme. Each row represents a target theme. The darker the box, the greater the similarity found between target and query. A perfect response would show a black box for every element along the main diagonal and white in all other positions.

The results of using the HMM and the Forward algorithm on this example are far from perfect. In fact, when the sixth theme in the database ("A Shot of Rhythm and Blues") is presented as a query, the HMM with the highest probability of generating it is the one created from theme nine ("Across the Universe").

In this test, the architecture we chose for the HMM works against the use of the Forward algorithm. Recall pitch is ignored by observation probability model 27. Pitch was not, however, ignored in determining the model transition architecture when HMMs were created from themes. Thus, branching points were created that reduce the probability of a particular path in the correct model. Given a sufficiently bad observation-probability model, these branching points can result in the right model generating a lower score than the wrong one.

To understand this better, consider Figure 11. Model A was generated from string A using a method similar to the one we described for making a Markov model from a sequence of ⟨*pitch interval, IOI*⟩ duples. Model B was generated from string B in the same way. During model construction, Greek letters (the hidden states) were assumed to have a one-to-one correspondence with Roman letters (the observations). This results in the two models as seen.

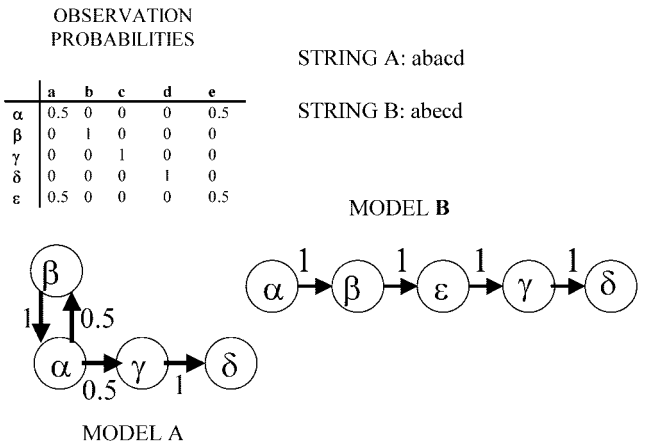

FIG. 11. Two HMMs.

The observation probability table in the figure describes a one-to-one correspondence between observation and hidden state, except for the observations "a" and "e." This table makes both hidden states α and ε equally likely to generate either of these two observations. Given this, the Forward algorithm on model A generates a score of $0.5 \cdot 0.5 \cdot 1 \cdot 1 \cdot 0.5 \cdot 0.5 \cdot 1 \cdot 1 \cdot 1 = 0.0625$, when passed string A as an observation sequence. Model B, however, generates a score of $0.5 \cdot 1 \cdot 1 \cdot 1 \cdot 0.5 \cdot 1 \cdot 1 \cdot 1 \cdot 1 = 0.25$. Thus, Model B would be returned as the most likely one, which is clearly the wrong result. This highlights the importance of choosing a good underlying architecture and ensuring the mapping from observation to hidden state does not undermine the descriptive power of the architecture.

Test 2: Observation Probability Models and the Forward Algorithm

Consider that 23 of the 29 models in Table 1 performed perfectly on Test 1. Which of these models allows the Forward algorithm to perform best when presented with real queries? Since the architecture for each HMM was fixed by the creation of the model, the only variation was the choice of observation probability model and the subset of the queries sung by the three people mentioned in an earlier section. Queries were broken down into sets by singer, resulting in three sets of queries and a total of $29 \cdot 3 = 87$ trials. We hypothesized the observation-probability model created from the analysis of a particular singer would result in the best performance on queries by that singer.

Figure 12 shows the mean reciprocal right rank (MRRR) for each combination of singer and observation-probability model. Recall that for MRRR on this data set, a perfect score is 1 and chance is $1/130 = 0.0077$. Scores below chance indicate a negative correlation between right rank and the rank generated by an algorithm. No trial resulted in a number below chance. The lowest scoring trial used observation-probability model 27 on Singer 3, with a MRRR of 0.012732 and a median right rank of 106.5. The best scoring trial was also for Singer 3 and used observation-probability model 26. For this trial, the MRRR was 0.68358 and the median right rank was 1.

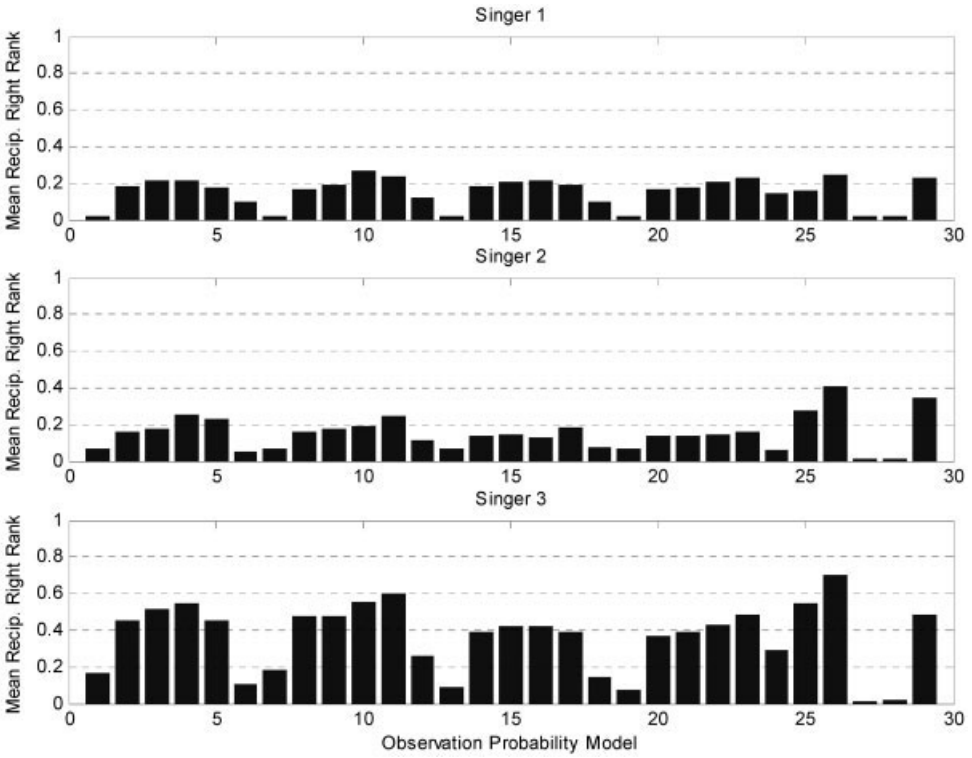

FIG. 12. Mean reciprocal right rank using the Forward algorithm.

TABLE 2. Best results: Forward algorithm.

Singer	Observation probability model	Mean recip right rank	Right answer in top 5	Right answer top choice	Number of queries
Singer 1	10	0.26	8 (29%)	6 (21%)	28
Singer 2	26	0.41	7 (41%)	6 (35%)	17
Singer 3	26	0.70	21 (75%)	19 (68%)	28

Table 2 shows the results of the best trial for each singer. Here, the number for the observation probability model is the index of the observation-probability model from Table 1 that generated the highest MRRR using the Forward algorithm. As expected, the highest performance for Singer 1 was achieved using model 10, an observation probability model based on Singer 1's actual data. Note that model 10 ignores rhythmic information; indicating inclusion of rhythmic information is actually detrimental to the performance of the matcher. This may be because the observation probability model for rhythm does not properly support the architecture of the model. An example of how this can occur was given in an earlier section.

Unexpectedly, the best performance on Singer 3's queries was not achieved using an observation probability model based on his data, but rather with the synthetic model 26. This model proved to be the best overall model for this algorithm, as it provided the best performance for Singer 3 and Singer 2 and second-best for Singer 1. Table 2 shows the numerical scores for the highest-scoring observation probability model for each of the three singers.

Test 3: String-Matcher Optimization

To optimize the string matcher, we fixed the query set to the queries used by the HMM. The observation probability models used were the 29 from Table 1.

Although there is a well-developed theoretical basis for creating the log-odds match probability table for string alignment, a method for choosing good skip-penalty values is not well developed. Not having a good model, we chose a brute-force solution to finding good penalty values. The query skip penalty was varied from 0 (skips cost nothing) to 12 (slightly more expensive than the worst log-odds match score possible). The theme skip penalty was varied similarly. This resulted in $13 \cdot 13 = 169$ combinations.

Finally, it seemed unlikely that queries and targets would match exactly in starting and endpoints. Thus, we decided to try both the Global and Local string-matching algorithms. This resulted in a total of 29 match score matrices, 169 skip-cost combinations and two match algorithms, for a total of 9,802 trials, where a single trial consists of the ranking of the entire target database for each of the queries in the query set by a ranker with fixed properties.

Table 3 contains the results for the best combination of skip costs and observation-probability model for each of the three singers on the Global-matching algorithm, as measured by mean reciprocal right rank (MRRR). One thing to note is that the optimal combination of skips' costs varies from singer to singer and search algorithm to search algorithm. The same also holds true for observation probability model. In all cases, the best observation probability model was a synthetic model, not based on actual singing data. Another thing to note is the target skip cost for the global matcher. A global matcher must account for every note in a target, whether or not there is a corresponding note in the query. In our data set, queries tend to be shorter than targets. It may very well be optimal to minimize the cost of skipping an element of the target in this situation. Thus, the cost of skipping a target element is reduced to 0 for two of the singers.

Table 4 contains the values for target skip cost, query skip cost, and observation probability model that optimize Local string-matcher performance for each singer's queries. As can be seen, the ideal target skip cost is no longer zero for any singer. Interestingly, the ideal relative cost of skipping a query element vs. skipping a target element varies greatly between singers from a three-to-one ratio all the way to a one-to-five ratio. For Singer 2, the cost of skipping a query element could be varied from 5 to 12 without affecting performance. Other singers showed a single best point setting for skip costs.

The winning observation probability models were again the synthetic ones, although not necessarily the same ones as were the optimal for the global matcher or the Forward algorithm. There is, however, a tendency for model 26 (exponential decay around the diagonal for *pitchInterval*, even probability for *IOIratio*) to appear as a top choice for

TABLE 3. Best results: Global string matcher.

Singer	Observation probability model	Target skip cost	Query skip cost	Mean reciprocal right rank	Right answer in top 5	Right answer top choice	Number of queries
Singer 1	26	0	1	0.35	13 (46%)	8 (29%)	28
Singer 2	25	1	2	0.25	5 (29%)	4 (24%)	17
Singer 3	25	0	2	0.43	12 (43%)	11 (39%)	28

TABLE 4. Best results: Local string matcher.

Singer	Observation probability model	Target skip cost	Query skip cost	Mean reciprocal right rank	Right answer in top 5	Right answer top choice	Number of queries
Singer 1	26	3	1	0.40	13 (46%)	10 (36%)	28
Singer 2	29	1	5 to 12	0.44	8 (47%)	7 (41%)	17
Singer 3	26	4	3	0.74	22 (79%)	20 (71%)	28

all three algorithms. This indicates the observation probability models we collected from sung data are inadequate, especially for rhythmic information.

A breakdown of system performance by query is enlightening. Figure 13 shows the ranking results for all Local string-matcher trials on Singer 2 using error model 26, the best error over-all error model when all three singers are taken into consideration. There were 169 such trials as the skip costs were varied for both query and target. Song names preceded by an asterisk indicate songs whose themes did not overlap any portion of the sung query. As expected, coverage errors result in poor performance, no matter what the skip cost. Several queries showed significant variation in pitch contour between the sung query and the stored theme. Among these was "All You Need is Love," where the person sang a harmony line, while the main melody line was stored in the target. In "Here Comes the Sun" the singer left out a number of notes, leading to a wide range in results as the skip costs are varied. The singer sang a monotonic portion of "Back in the USSR," leaving only rhythm to identify the song. Unfortunately, error model 26 is insensitive to rhythm and results in poor performance on this query.

Reality Check: Human Recognition Rates

We define the recognition rate for a system to be the percentage of queries where the correct target was chosen as the top pick. The highest recognition rate achieved by any of our systems was 71% by the local-string matcher on Singer 3. The lowest rate was 21% by the Forward algorithm on Singer 1. Without some kind of baseline, it is difficult to put these figures into perspective. We decided to create a baseline by presenting the sung queries to the singers who generated the query set to see how many of them would be recognized.

Two months after the queries were made, the three singers were gathered into a room and presented the original recordings of the queries in a random order. Each recording was presented once, in its entirety. Singers were told that each one was a sung excerpt of a piece of music performed by the Beatles and that the task was to write down the name of the Beatles song the person in the recording was trying to sing. Only one answer was allowed per song and singers were given a few (no more than 15) seconds after each query to write down an answer.

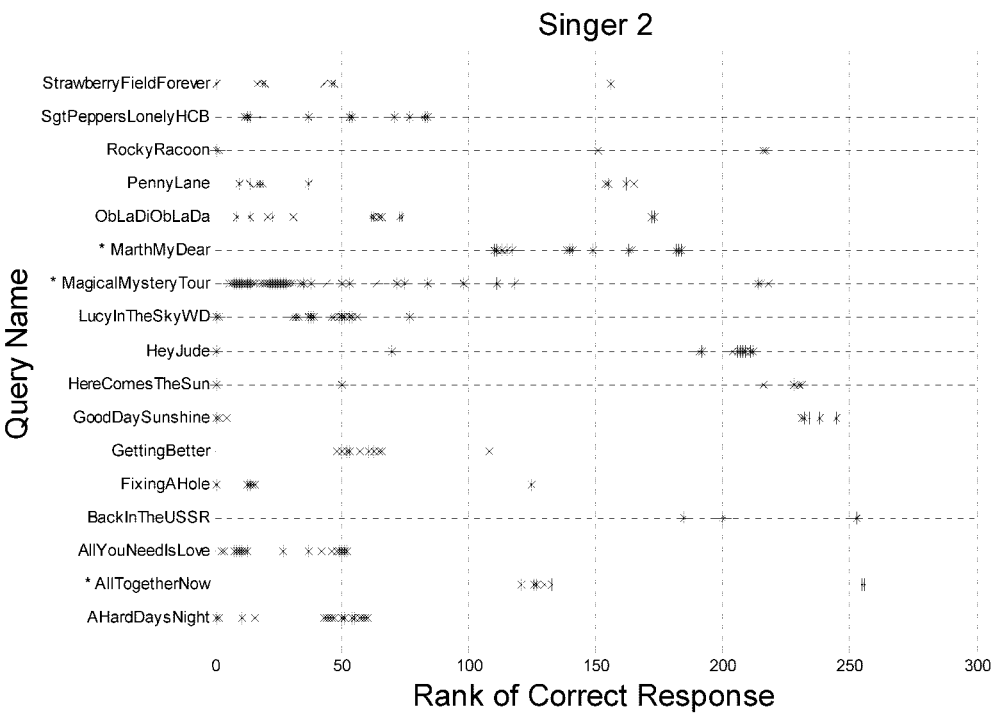

FIG. 13. Rank of right response, Singer 2. Error model 26: Local string matcher.

TABLE 5. Human performance versus machine performance.

	Identified by							
Sung by	Singer 1	Singer 2	Singer 3	Other 2 singers	Forward	Global	Local	*N*
Singer 1	27 (96%)	14 (50%)	20 (71%)	61%	21%	29%	36%	28
Singer 2	12 (71%)	14 (82%)	13 (76%)	74%	35%	24%	41%	17
Singer 3	22 (79%)	13 (46%)	25 (89%)	63%	68%	39%	71%	28
			Average	66%	41%	31%	49%	

Once all queries had been heard, responses were graded. Recall that queries were sung with nonsense syllables and that lyrics were not used. Because of this, we judged any response that contained a portion of the correct title or a quote of the lyrics of the song as a correct answer. All other answers were considered to be wrong.

Table 5 contains the results of the human trials, along with the results for the machine song recognition algorithms, where credit was received only for selecting the right song as the top choice. Each row contains the recognition rates for queries sung by a particular singer. Each column identifies the recognition rates when using a particular method for identifying sung queries. The column labeled "Other 2 Singers" in Table 5 contains the average recognition rates of the two singers who did NOT sing a particular set of queries. Thus, for Singer 2's queries, the "Other 2 Singers" value is the average of how well Singer 1 and Singer 3 recognized Singer 2's queries.

It is interesting to note that the human listeners achieved an average recognition rate of only 66%, when presented with queries sung by another person. While there are many possible explanations, this figure was lower than expected and may provide a rough estimate as to how well one can expect a machine system to do. Even more interesting was the inability of Singers 2 and 3, both with graduate degrees in music performance, to achieve even a 90% recognition rate on their own sung queries, while Singer 1 achieved a much higher recognition rate on his own queries.

Conclusions

Mismatch between a transcribed query and the target musical theme in the database is common, and must be handled by any QBH system. Error models that allow graceful degradation in the face of inexact queries by explicit error modeling outperform models that assume no singer (or transcriber) error in our trials. Interestingly, synthetic error models based on exponential decay provided comparable performance to data-derived error models designed to handle the errors of a particular singer. This may mean we can avoid time-consuming training of error models for individuals by using an appropriate default generic error model.

Since the best performing error model was a synthetic one that ignored rhythm, future work involving real user data must collect data in a way that more accurately captures the kinds of errors made in the course of singing than was done in our study. This should have a significant impact on the performance of QBH systems, especially when attempting to correctly match rhythmic patterns.

The experimental results also show that on this data set, the string-matching systems perform slightly better than the HMM approaches, with neither performing as well as humans. The string matchers also ran roughly 12 times faster than the HMM classifier on the given set of queries and targets, with the string matchers taking an average of 0.16 seconds per query and the HMM taking 1.97 seconds per query. We conclude that the HMM architecture chosen in our experiment (allowing loops), can have a significant disadvantage for certain error classes. At this point, we believe that the left-right approach, as used by the string approaches, is superior to the "loop" model used by our HMM. We have begun to study a comprehensive left–right HMM architecture that accounts for various skips and insertions and are preparing more extensive experiments, using a larger database and many more singers, to compare this architecture to the string matchers.

Acknowledgments

We gratefully acknowledge the support of the National Science Foundation under grant IIS-0085945, and The University of Michigan College of Engineering seed grant to the MusEn project. The opinions in this article are solely those of the authors and do not necessarily reflect the opinions of the funding agencies. We also thank Roger Dannenberg for many helpful comments.

References

Birmingham, W., Pardo, B., et al. (2002). The MusArt music-retrieval system: An overview. D-lib Magazine, 8. Available: http://www.dlib.org

Clarisse, L.P., Martens, J.P., et al. (2002). An auditory model based transcriber of singing sequences. Paper presented at ISMIR 2002, The 3rd International Conference on Music Information Retrieval, Paris, France.

Clausen, M., Englebrecht, R., et al. (2002). Proms: A Web-based tool for searching in polyphonic music. Paper presented at the International Symposium on Music Information Retrieval, Paris, France, October.

Dannenberg, R. (1984). An on-line algorithm for real-time accompaniment. Paper presented at the International Computer Music Conferences, Paris: France.

Doraisamy, S., & Ruger, S. (2002). A comparative and fault-tolerance study of the use of N-grams with polyphonic music. Paper presented at

ISMIR 2002, The 3rd International Conference on Music Information Retrieval, Paris, France, October 2002.
Downie, S., & Nelson, M. (2000). Evaluation of a simple and effective music information retrieval method. Paper presented at the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Athens, Greece, July 24–28, 2000.
Durbin, R., Eddy, S., et al. (1998). Biological sequence analysis, probabilistic models of proteins and nucleic acids. Cambridge, UK: Cambridge University Press.
Durey, A.S., & Clements, M. (2001). Melody spotting using hidden Markov models. Paper presented at the International Symposium on Music Information Retrieval, Bloomington, IN, October, 2001.
Gotoh, O. (1982). An improved algorithm for matching biological sequences. Journal of Molecular Biology, 162, 705–708.
Hoos, H., Rentz, K., et al. (2001). GUIDO/MIR—An experimental musical information retrieval system based on GUIDO music notation. Paper presented at the International Symposium on Music Information Retrieval, Bloomington, IN, October, 2001.
Hu, N., Dannenberg, R., et al. (2002). A probabilistic model of melodic similarity. Paper presented at the International Computer Music Conference (ICMC), Goteborg, Sweden, September 16–21, 2002.
Kornstadt, A. (1998). Themefinder: A Web-based melodic search tool. In Melodic similarity concepts, procedures, and applications. In W. Hewlett & E. Selfridge-Field (Eds.), Cambridge, MA: MIT Press.
McNab, R.J., Smith, L.A., et al. (1996). Towards the digital music library: Tune retrieval from acoustic input. Proceedings of the first ACM International Conference on Digital Libraries. Bethesda, MD, March 20–23, 1996.
Meek, C., & Birmingham, W.P. (2002). Johnny can't sing: A comprehensive error model for sung music queries. Paper presented at ISMIR 2002, The 3rd International Conference on Music Information Retrieval, Paris, France, October, 2002.
MIDI Manufacturers Association. (1996). The Complete MIDI 1.0 Detailed Specification. Los Angeles, CA: Author.
Needleman, S.B., & Wunsch, C.D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48, 443–453.
Pardo, B., & Birmingham, W.P. (2001). Following a musical performance from a partially specified score. Paper presented at the Multimedia Technology Applications Conference, Irvine, CA, February 8–10, 2001.
Pardo, B., & Birmingham, W. (2002). Improved score following for acoustic performances. Paper presented at the International Computer Music Conference (ICMC), Goteborg, Sweden.
Pardo, B., & Birmingham, W.P. (2002). Encoding timing information for musical query matching. Paper presented at ISMIR 2002, 3rd International Conference on Music Information Retrieval, Paris, France, October, 2002.
Pickens, J. (2000). A comparison of language modeling and probabilistic text information retrieval. Paper presented at the International Symposium on Music Information Retrieval, Plymouth, MA, October, 2000.
Puckette, M., & Lippe, C. (1992). Score following in practice. Paper presented at International Computer Music Conference, San Jose, CA.
Rabiner, L., & Juang, B.-H. (1993). Fundamentals of speech recognition. Englewood Cliffs, NJ: Prentice-Hall.
Raphael, C. (1999). Automatic segmentation of acoustic musical signals using hidden Markov models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(4), 360–370.
Shifrin, J., Pardo, B., et al. (2002). HMM-based musical query retrieval. Paper presented at the Joint Conference on Digital Libraries, Portland, Oregon.
Sterian, A. (1999). Model-based segmentation of time-frequency Images for musical transcription. Ann Arbor, MI: University of Michigan.
Tolonen, T., & Karjalainen, M. (2000). A computationally efficient multipitch analysis model. IEEE Transactions on Speech and Audio Processing, 8(6), 708–716.
Tseng, Y.H. (1999). Content-based retrieval for music collections. Paper presented at the ACM International Conference on Information Retrieval (SIGIR), Berkeley, CA, August 15–19, 1999.
Uitdenbogerd, A., & Zobel, J. (1999). Melodic matching techniques for large music databases. Paper presented at the Seventh ACM International Conference on Multimedia, Orlando, FL, October 30–November 5, 1999.

Appendix: Explicit Rhythm Representation Using *IOIratios*

Melody matching has, in general, concentrated on matching pitch contour. Typically, rhythm is encoded implicitly. The number of contiguous, fixed-duration frames with the same pitch indicates note length. Figure A1 shows three sequences as a piano roll. Vertical lines indicate the boundaries of fixed-duration frames. A letter above each frame indicates the pitch class present in that frame.

Given an implicit representation of timing information, music matchers have difficulty in dealing with tempo scaling. Looking at Figure A1, it is clear that S_1 and S_2 represent the same melody performed at two different speeds and S_3, while containing all the notes in S_1, is another melody entirely. Unfortunately, for many string-alignment style matchers, the least-cost alignment between S_1 and S_2 is the same as the least-cost alignment between S_1 and S_3, making it impossible for the matcher to determine which two melodies are the most closely related. Clearly, this is not a desirable result.

Similarity between strings can be measured by the likelihood both were generated by the same Markov model (Markov models are described in this article in the section titled, "Themes as Markov Models"). In this case, the strings are considered observation sequences and the Forward algorithm is used to determine how likely a given model is to generate each string. Given a set of models, two strings are deemed to be of the same class if both are most likely generated by the same model.

Model A in Figure A2 illustrates a Markov model capable of generating either S_1 or S_2, but not S_3. Here, states are labeled with the letter they emit and transition probabilities are indicated by arrows between states, labeled by values between 0 and 1. The model will generate either one or two Gs, anywhere from one to infinite As and either one or two Bs.

This model illustrates the two basic approaches to handling timing variation by varying the structure of a Markov model: self-loops (the state that generates the As), and repeated states (the Gs and Bs). Both approaches have their drawbacks. Repeated states allow, at most, a fixed number of repetitions. For example, a sequence, S_4 = GGGGAAAABBBB could not be generated by the model depicted in Figure A2. Self-loops permit an infinite number of repetitions, but the probability of generating a sequence of

FIG. A1. Encoding of timing information.

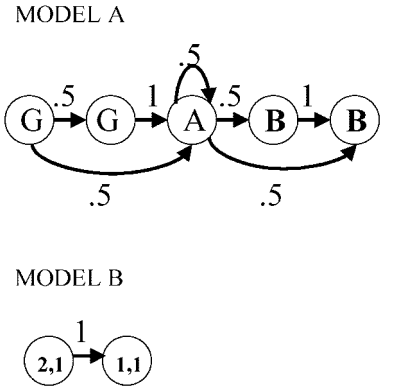

FIG. A2. Two Markov models.

repetitions decays exponentially with the length of the sequence. A model using three self loops, one for G, one for A, and one for B would be less likely to generate S_4than S_2, with S_1 being more likely than either S_4 or S_2.

A better way to deal with such variation is to change how rhythm is encoded, taking an example from pitch contour. Consider the problem of aligning the song, "Happy Birthday" in F to "Happy Birthday" in C. If both melodies are encoded using absolute pitch, the resulting strings look quite different. If they are encoded as change in pitch relative to the previous note (*pitchInterval*), the pitch contours are identical.

A typical measure used for timing in music applications is the inter onset interval between note events. This is the interval between the onset of note i and the onset of note $i + 1$. Timing information is generally measured in absolute units, such as milliseconds. This may be normalized by dividing all *IOI*s in a given performance by a reference duration, d_{ref}.

For a performed passage, such as a sung query to a music theme-finder, d_{ref} would ideally be the duration of a beat, as it is in written notation. Unfortunately, it is not always clear what the beat is in a query and beat tracking on a relatively short excerpt (10–15 notes) may not always be effective. Thus, another basis for d_{ref} must be found. A simple substitute for the beat is the duration of the previous event This is expressed in Equation A1.

$$IOIratio_n = \frac{IOI_n}{IOI_{n+1}} \tag{A1}$$

Representing the melodies in the figure as sequences of ⟨*pitchInterval, IOIratio*⟩ duples brings out the similarity between the first two melodies. In fact, using this representation they are identical.

Model B in Figure A2 contains a Markov model capable of generating both S_1 and S_2 with equal probability. In this figure, states represent ⟨*pitchInterval, IOIratio*⟩ duples. This model illustrates the effect a change of encoding has upon the Markov model needed to generate the sequences of interest.