HMM-Based Musical Query Retrieval

Jonah Shifrin, Bryan Pardo, Colin Meek, William Birmingham
EECS Dept, University of Michigan
110 ATL, 1101 Beal Avenue
Ann Arbor, MI 48109-2110
+1 (734) 936-1590
jshifrin@umich.edu, bryanp@umich.edu, meek@umich.edu, wpb@eecs.umich.edu

ABSTRACT
We have created a system for music search and retrieval. A user sings a theme from the desired piece of music. Pieces in the database are represented as hidden Markov models (HMMs). The query is treated as an observation sequence and a piece is judged similar to the query if its HMM has a high likelihood of generating the query. The top pieces are returned to the user in rank-order. This paper reports the basic approach for the construction of the target database of themes, encoding and transcription of user queries, and the results of initial experimentation with a small set of sung queries.

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

General Terms
Algorithms

Keywords
music, hidden Markov model, melody, Forward algorithm, database

1. INTRODUCTION
Our group has created a system for music search and retrieval called MuseArt [1]. 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 system transcribes the sung query and searches for related themes in a database, returning the most similar themes, given some measure of similarity. We call this "retrieval by query." Retrieval by query for music has been investigated by several research groups [2][3][4] in recent years, typically with an emphasis on string matching techniques.

The system described in this paper assumes a matching based solely on timing and pitch contour. We represent each theme using a hidden Markov model (HMM). 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. A piece of music is deemed a good match if at least one theme from that piece is similar to the query. The pieces are returned to the user in order, ranked by similarity.

Other researchers [5] are also investigating the use of Markov models for music retrieval, but do not use hidden Markov models, forcing the system to require exact matches between query and theme. Our use of HMMs lets the system gracefully handle queries that contain errors. We consider this a strength of our approach. Our state representation also makes the system robust to differences in the tempo and transposition of the query, as compared with Durey[6], who relies on the target and query being presented at the same tempo and in the same pitch range.

JCDL '02, July 13-17, 2002, Portland, Oregon, USA.

2. 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-based representation using a pitch-tracking system developed at Carnegie Mellon University [7] based on an enhanced autocorrelation algorithm [8].

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). Pitch tracking can be thought of as the equivalent of character recognition in the text world.

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.

Figure 1 shows a time-amplitude representation of a sung query, along with example pitch-tracker output (shown as piano roll) and

a sequence of values derived from the MIDI representation (the *deltaPitch*, *IOI* and *IOIratio* values). Time values in the figure are rounded to the nearest 100 milliseconds.

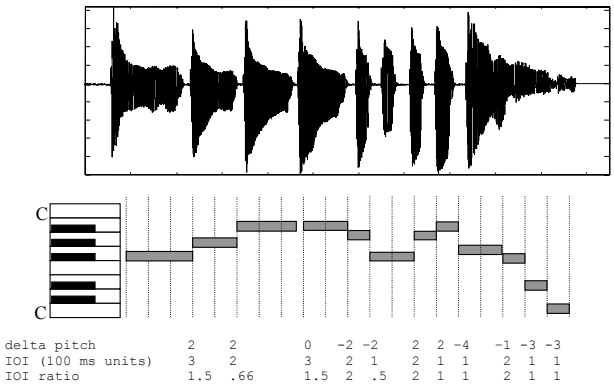

Figure 1: A sung query

We define the following.

- A *note transition* between note *n* and note *n*+1 is described by the duple <*deltaPitch*, *IOIratio*>.
- $deltaPitch_n$ 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, $IOI_n = IOI_n / duration_{n+1}$.

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. The *deltaPitch* component of a note transition captures pitch-change information. Two versions of a piece played in two different keys have the same *deltaPitch* values. The *IOIratio* represents the rhythmic component of a piece. This remains constant even when two performances are played at very different speeds, as long as relative durations within each performance remain the same.

In order to reduce the number of possible IOI ratios, we quantize them to one of 27 values, spaced evenly on a logarithmic scale. A logarithmic scale was selected because data from a pilot study indicated that sung *IOIratio* values fall naturally into evenly spaced bins in the log domain.

3. THEMES 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 2 represents a Markov model of a scalar passage of music. States are note transitions. Nodes represent states. The numerical value below each state indicates the probability a traversal of the graph will begin in that 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.

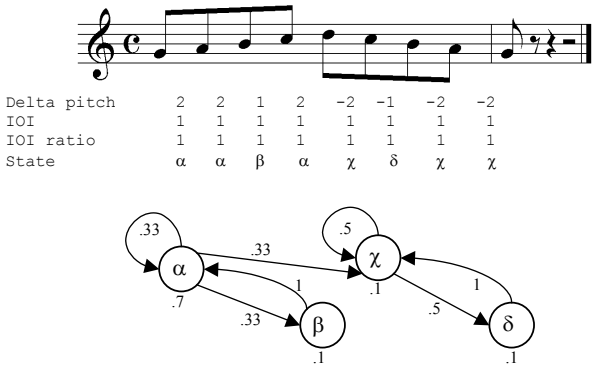

Figure 2: Markov model for a scalar passage

A model that explicitly maintains a probability distribution over the set of possible observations for each state is 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 of possible observations, $O = \{o_1, o_2, o_3, \ldots, o_n\}$.
- A probability distribution over the set of observations for each state in S.

In our approach, a query is a sequence of observations. Each observation is a note-transition duple, <*deltaPitch, IOIratio*>. Musical themes are represented as hidden Markov models.

3.1 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 2 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. Figure 3 is an example of this, where only a single state has two possible transitions, one back to itself and the other on to the next state.

The probability distribution for the initial state in each model in our database is given by the formula in Equation 1. Here, $|S|$ is the number of states in the model, p is a probability, and *first* is the state based on the first observation in the sequence from which the model was constructed. In the case of Figure 2, α is the *first* state.

$$\pi_i = \begin{cases} p, & \text{if } s_i = first \\ \dfrac{1-p}{|S|-1}, & \text{otherwise} \end{cases} \qquad \text{Equation 1}$$

If the themes in our database perfectly reflect the segments of a song that the subject will sing, we can simply set $p = 1$. We relax this restriction by setting $p<1$ and assigning a uniform distribution over the remaining states. This allows a traversal of the Markov Model for a theme to begin in any state with some low probability. This is done both because the database themes are sometimes poorly delimited and because the subject may reasonably choose to begin the query at a different point.

The appropriate value for p depends on the situation and must be approximated given a particular set of queriers and themes in the database. For the examples in Figure 2 and Figure 3, p has been set to 0.7.

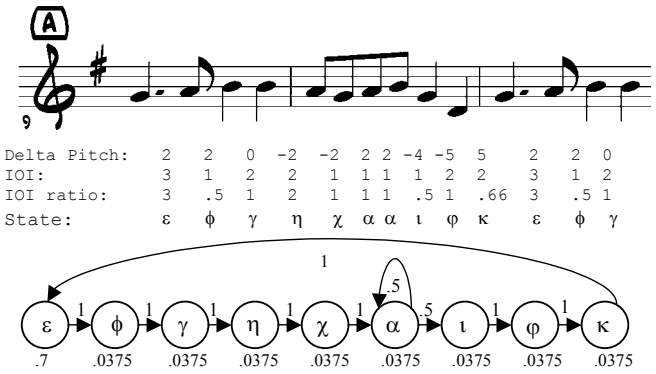

Figure 3: Markov model for *Alouette* fragment

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, *<deltaPitch, IOIratio>*. Given this, the observation sequence $q = \{(2,1), (2,1), (2,1)\}$ may be generated by the HMM in Figure 2 or the HMM in Figure 3.

3.2 Estimating Observation Probabilities

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 set of possible observations , *O*, given a hidden state, *s*.

Observations consist of duples *<deltaPitch, IOIratio>*. There are twelve 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 *deltaPitch* values. We quantize *IOIratio* to one of 27 values. This means there are 25*27= 675 possible observations given a hidden state. Since hidden states are also characterized by *<deltaPitch, IOIratio>*, there are 675 possible hidden states for whom observation probabilities need to be determined. The resulting table has 675^2, or over 450,000 entries.

We use a typical method to estimate probabilities. A number of observation-hidden state pairs are collected into a training set 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 entered.

$$P(o_i \mid s) = \frac{count(o_i, s)}{\sum_{j=1}^{|O|} count(o_j, s)} \qquad \text{Equation 2}$$

Creating a dataset of paired observations and hidden states from which to estimate nearly half a million observation probabilities is daunting. This can be made more tractable by assuming conditional independence between *deltaPitch* and *IOIratio*.

Given independence, two separate observation probability tables may be maintained, one for *deltaPitch* and one for *IOIratio.* The former having 25*25 = 625 values, the latter having 27*27=729 values. The probability of encountering any observation duple, given a hidden state duple, can then be derived from the two tables using Equation 3.

$$P(o \mid s) = P(deltaPitch_o \mid deltaPitch_s) * P(IOIratio_o \mid IOIratio_s) \qquad \text{Equation 3}$$

Even given the reduced size of the observation probability tables allowed by Equation 3, 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 again equal to one.

Given probability tables for *IOIratio* and *deltaPitch*, a Markov model constructed from a MIDI file, such as the one in Figure 3, may be treated as a hidden Markov model. A query is an observation sequence and the probability of an observation, given a state may be calculated using Equation 3.

4. 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.

4.1 The Forward Algorithm

The Forward algorithm [9], 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 maximum path length, L, the algorithm takes all paths through the model of up to L 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 $|S|^2L$ steps to compute the probability, where $|S|$ is the number of states in the model.

4.2 Selecting The Most Likely Model

Let there be an observation sequence (query), O, and a set of models (themes), M. An order may be imposed on 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 take this rank order to be a direct measure of the relative similarity between a theme and a query. Thus, the first theme is the one most similar to the query.

4.3 Removing Bias in the Forward Algorithm

We have found that, in practice, the value returned by the Forward algorithm is negatively correlated with the number of states in the model. This is an effect of model topology and the probability distribution used to determine the starting state for a model.

Many thematic models, such as that in Figure 3, are essentially deterministic. For a deterministic model, the factors that determine in the value returned by the Forward algorithm are the observation probabilities and the initial state distribution, as determined by Equation 1.

Equation 1 reduces the probability of starting in any state (except for the *first* state) as the number of states increases. This reduction is linear with the number of states in the model.

Assume that for observation sequence O, the highest probability path through a model, m, does not start in the *first* state. If a new state is added to m, the probability of this path will be reduced. This reduction tends to be greater than the probability of a path going through the new state. The effect is to lower the score of models with more states.

We offset this bias by introducing a scaling factor that varies linearly with respect to the number of states in the model, $|S|$, according to Equation 4, where a and b are constants.

$$k = a\,|S| + b \qquad \text{Equation 4}$$

An individual scaling factor is found for each HMM in the set of models (themes), M. When ranking themes for similarity to the observation sequence, the result of the Forward algorithm for each model is multiplied by its scaling factor and the scaled values are used to order the themes as described in the previous section.

5. EXPERIMENTAL RESULTS

As an initial test of the ideas in this paper, we constructed a database of pieces represented as hidden Markov models and generated a set of queries, sung by the authors. Themes in the database were ranked for similarity to each query and the ranked results were returned.

5.1 Baseline Matcher

In order to compare our results to a clear, easy-to-understand baseline, we implemented a simple string matcher that measures the edit distance between a potential target and a query. Allowable operations are a skip of a target element (deletion), a skip of a query element (insertion) and an alignment of the two elements. A skip of either a target or query element costs one point. An exact match between a target and query element is rewarded by a point. An inexact match costs one point. Themes are ranked by the cost of the best global alignment of each theme with the query. Matches were performed on *deltaPitch* information. The baseline matcher used no duration information.

5.2 Target Corpus Construction

We collected a corpus of 277 pieces of music encoded as MIDI from public domain sites on the web. The corpus contains a wide variety of genres, including Classical, Broadway show tunes, Jazz and popular music from the past 40 years. Pieces were selected on basis of familiarity to a wide audience of college students, as estimated by members of the research group. Each piece in the corpus was represented in a database by a set of *themes*, or representative monophonic melodic fragments. Themes were extracted automatically from each piece by an early version of MME, an automatic melodic motive extractor [10]. An average of 9.58 themes per piece were found by MME, resulting in a database of 2653 monophonic themes.

MME is designed to do the musical equivalent of keyword identification. It identifies all patterns, characterized by melodic contour (or interval sequence), in a piece and then uses a scoring mechanism to determine the relative importance of these patterns. It then outputs a series of "themes" corresponding to the highest scoring patterns.

Once each theme was extracted and placed in an individual MIDI file, the HMM for that theme was generated automatically, and placed in the database. Each theme was then indexed by the piece it was derived from.

In order to get a measure of how confusable the individual themes are, we randomly selected 100 of the 2653 MIDI themes used to generate the Markov models in the database. These files were transformed directly into sequences of *<deltaPitch, IOIratio>* tuples and passed to both the baseline and the HMM-based query matching systems. Since there is no transcription error for such queries and queries are guaranteed to be in the database, the only source of error is system inability to distinguish between themes in the database.

The HMM-based query system returned the correct theme as the top match in 97% of cases. The baseline string matcher returned the correct theme as the top match in 91% of cases. This indicates that, in general, the themes in the database are quite distinguishable.

5.3 Query Corpus Construction

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.

Four male singers (the authors of this paper) generated queries for the experiment. Two of the singers have graduate degrees in instrumental (not vocal) music performance. The remaining singers have no musical training beyond private instrumental lessons. None are trained as singers. All are between the ages of twenty and forty and are North American native speakers of English.

Sung queries were recorded in 8 bit, 22.5 kHz mono using an Audio-Technica AT822 microphone from a distance of roughly six inches. Recordings were made directly to an IBM ThinkPad T21 laptop 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.

Each singer was asked to sing three well-known pieces from the target corpus: *America the Beautiful*, Queen's *Another One Bites the Dust*, and The Beatles' *Here Comes the Sun*. Each singer was asked to sing any portion of the melody he considered significant. No tempo or key was specified and singers were allowed to go on as long as desired. After each query, the singer had the option of singing the song again, or submitting the query. Only the final submitted query for each song was used.

Once the required three songs were sung, each singer was asked to sing an additional three songs from the list of 277 pieces in the target corpus, using the same protocol as for the required list of songs. Recordings of all queries were stored by song title, for the purpose of testing system performance, given known correct answers.

The resulting query corpus contained six queries by each of four singers, for a total of 24 queries representing 15 different pieces.

5.4 Ranking Results

For each query, the full database of 2653 themes was scored using the Forward algorithm on the HMM representing each theme. Scores were then scaled in accordance with Equation 4. Each of the 277 pieces in the target corpus was represented in the database by a set of roughly nine automatically generated themes. Pieces were ranked in order by the score of their highest-ranking theme. Each theme was also scored by the baseline string matcher, with both query and theme represented as a sequence of *deltaPitch* values. As with the Forward algorithm, pieces were ranked in order by the score of their highest-ranking theme.

Table 1 shows the rank of the correct answer, broken down by rank scoring method. The table shows HMM-based ranking using the Forward algorithm clearly outperforms ranking based on edit-distance with the simple string matcher. In fact, the HMM approach placed the correct answer in the top five queries three times as often as the string matcher did and outscored the string matcher in twenty out of twenty-four cases. The median rank of the correct answer was 4th with the HMM approach and 49th with the string matcher. This is echoed by the mean difference in ranking reported by the two systems when compared on the same query, with the HMM system ranking the correct piece an average of 44.1 places higher than the string matcher.

Table 1: Number of cases by rank of correct answer

System	HMM		String Matcher	
Rank of Correct Answer	Number of Cases	Cumulative Percentage	Number of Cases	Cumulative Percentage
1	10	41.7%	4	16.7%
2 to 5	4	58.3%	1	20.8%
6 to 10	0	58.3%	1	25.0%
11 to 25	3	70.8%	2	33.3%
26 to 50	1	75.0%	4	50.0%
51 to100	3	87.5%	4	66.7%
Over 100	3	100.0%	8	100.0%

Figure 4 shows rankings returned by the HMM-based system for all queries, broken down by singer and piece sung. Lower numbers indicate better results. As the figure shows, rank scores vary tremendously by piece and singer.

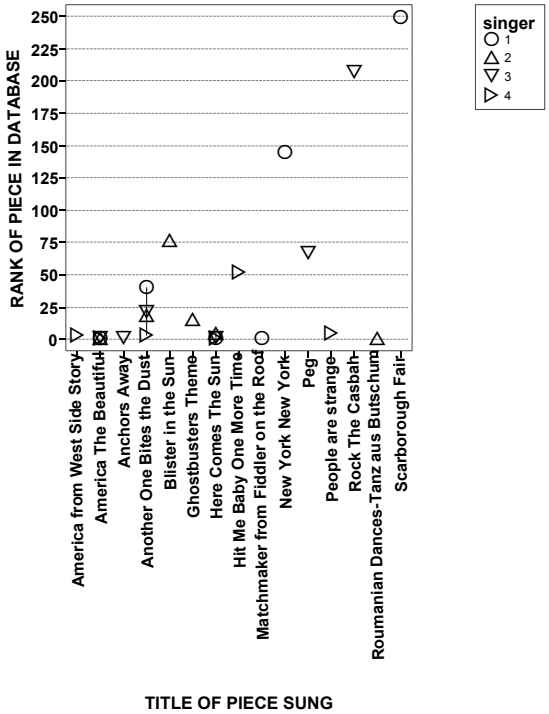

Figure 4: Rank of correct answer by piece and singer

A query was defined as a success when the correct piece was returned as one of the top five matches. Study of the ten cases where the correct title received a score of sixth or worse by the HMM system revealed three main sources of system error: pitch-tracker error (five cases), database coverage (one case), and ranking system error (three cases).

Singer 2's *Ghostbusters* query is an example of pitch-tracker error. While the initial .wav file was recognizable as the intended piece, the output of the pitch tracker had so many note skips, that

it was difficult to recognize when played as MIDI, resulting in a ranking of 16th by the HMM system and 64th by the string match system.

The pitch tracker was successful with Singer 1's *Scarborough Fair*. In this case, the database did not contain a model covering the portion of the song Singer 1 performed in the in the query. As a result, the HMM system ranked it 249th and the string matcher ranked it 273rd.

Ranking errors occur when the transcribed query sounds recognizably similar to a theme in the database, but the retrieval program gives the song a low ranking. The transcription of Singer 1's *New York New York* had a few note skips, but otherwise sounded recognizably similar to one of the *New York New York* themes in the database. The HMM system, however, ranked the piece 145th. The string matcher did a better job on Singer 1's *New York New York*, ranking it 44th.

The query for *Rockin' the Casbah* illustrates a situation no system is likely to cope with successfully. The original recording of this query was unrecognizable when played for several people familiar with the piece. It appears the singer had poor recall of the melody. Accordingly, the HMM system ranked it 207th, and the string matcher ranked it 234th.

Successful queries, such as *America the Beautiful*, tended to have held notes and to be sung more slowly. We attribute this to the cleaner output of the pitch tracker and note segmenter with such queries.

6. CONCLUSIONS

We have described a system for retrieving pieces of music from a database on basis of a sung query. The database is constructed automatically from a set of MIDI files, with no need for human intervention. Pieces in the database are represented as hidden Markov models (HMMs) whose states are note transitions. Queries are treated as observation sequences and pieces are ranked for relevance by the Forward algorithm. The use of note transitions as states and the Hidden Markov approach make for a system that is relatively robust in the face of key and tempo change. The use of observation probability distributions for hidden states deals with systematic error in query transcription.

Problems in our model include poor handling of queries longer than the maximum length path through the HMM for a theme (i.e., when the query quotes a longer segment of the piece than is in the database) and queries that skip notes. The addition of low probability "short cut" connections in the HMMs may alleviate the note-skip problem. Long queries may be handled by windowing the query to a size no longer than the longest path in the HMM under consideration.

A goal for the next version of our system is a better observation probability training set that includes a larger set of hidden-state-observation pairs from multiple singers. This should translate into a more singer-independent system and one that does a better job in dealing with pitch tracker error.

Hidden Markov models provide an excellent tool for modeling music queries. The results of our experiments with this "first-step" implementation indicate both the promise of these techniques and the need for further refinement. Refinements to the hidden model topology and of the observation model will allow us to model a broader range of query behavior, and improve the performance of the system.

7. 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 paper 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.

8. REFERENCES

[1] Birmingham, W.P., Dannenberg, R.D., Wakefield, G.H., Bartsch, M., Bykowski, D., Mazzoni, D., Meek, C., Mellody, M., Rand, W. Musart: Music Retrieval Via Aural Queries, in Proceedings of ISMIR 2001 (Bloomington, IN, October 2001), 73-81

[2] McNab, R. J., Smith, L. A. et al. Towards the digital music library: tune retrieval from acoustic input. Digital Libraries, ACM. 1996.

[3] Clausen, M., Englebrecht, R. et al. Proms: A web-based tool for searching in polyphonic music. Proceedings of the International Symposium on Music Information Retrieval, 2000.

[4] Tseng, Y. H. (1999). Content-based retrieval for music collections. SIGIR, ACM. 1999.

[5] Hoos, H., Rentz, K., Gorg, M. GUIDO/MIR – an Experimental Musical Information Retrieval System based on GUIDO Music Notation, in , in Proceedings of ISMIR 2001 (Bloomington, IN, October 2001), 41-50

[6] Durey, A., Clements, M. Melody Spotting Using Hidden Markov Models, in Proceedings of ISMIR 2001 (Bloomington, IN, October 2001)

[7] Mazzoni, D. and Dannenberg R.D. Melody Matching Directly From Audio, in Proceedings of ISMIR 2001 (Bloomington, IN, October 2001), 17-18

[8] Tolonen, T. and Karjalainen, M. A computationally efficient multi-pitch analysis model. IEEE Transactions on Speech and Audio Processing, Vol. 8, No. 6, Nov. 2000.

[9] Rabiner, L. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceeedings of the IEEE. Vol. 77, No. 2, 1989, 257-286.

[10] Meek, C., Birmingham, W. Thematic Extractor, in Proceedings of ISMIR 2001 (Bloomington, IN, October 2001), 119-128.