

Indexing Hidden Markov Models for Music Retrieval

Hui Jin

Department of Electrical Engineering and
Computer Science
The University of Michigan
3334 EECS Building 1301 Beal Ave
Ann Arbor, MI 49109-2122
(10)(734)763-5243
jjinh@eecs.umich.edu

H. V. Jagadish

Department of Electrical Engineering and
Computer Science
The University of Michigan
2238 EECS Building 1301 Beal Ave
Ann Arbor, MI 48109-2122
(10)(734)763-4079
jag@eecs.umich.edu

ABSTRACT

Hidden Markov Models (HMMs) have been suggested as an effective technique to represent music. Given a collection of musical pieces, each represented by its HMM, and a query, the retrieval task reduces to finding HMM most likely to have generated the query. The musical piece represented by this HMM is frequently the one rendered by the user, possibly imperfectly.

This method might be inefficient if there is a very large music database, since each HMM to be tested requires the evaluation of a dynamic programming algorithm. In this paper, we propose an indexing mechanism that can aggressively prune the set of candidate HMMs to be evaluated in response to a query. Our experiments on a music database showed an average of a seven-fold speed up with no false dismissals.

1. INTRODUCTION

Music retrieval is the process of retrieving expected songs from a music database. When a composer creates a melody, she might want to find similar ones created before. Music stores and music librarians are often asked to find a piece of music based on a few sung or hummed notes. This task, music retrieval by content, is challenging considering the fact that there is a large amount of music. For example, the U.S. Library of Congress holds over six million pieces of sheet music and the National Library of France has 300,000 hours of sound recording, 90% of it music [2]. To make it even more difficult, music is an art form, how similar two pieces of music is subjective. Thus, music retrieval has been a hot research topic recently.

The techniques applied in music retrieval should be both effective and efficient. Effective means that the result returned by a system should be the one a user expects, i.e. the one most similar to the query. Efficiency means that the system should not let a user wait too long for the result. Several proposals in the literature have explored Hidden Markov Model (HMM) to facilitate music retrieval. These techniques have been shown to be effective [16]. However, these techniques are not very efficient. Using these techniques, a query has to be compared with each piece of HMM. There is a need to index the HMMs to speed up the retrieval when faced with large music databases.

We propose a simple yet efficient mechanism to index the HMMs. In this specific HMM we consider, each state is rep-

resented by an interval and inter onset interval ratio. Each transition in an HMM is turned into a four dimensional box. Other HMM definition can be handled similarly. The box obtained may have more than four dimensions. All the boxes are inserted into one R*-tree [15], an indexing structure for multi-dimensional data. When a query is submitted, each transition of the query is first converted into a four dimensional point and the boxes in the R*-tree that contain these points are then found. The HMMs are ranked by the number of those boxes and their corresponding songs are taken as similar songs. After the candidate HMMs are selected, we use common technology, the forward algorithm, to identify the most similar song.

We have implemented this design and done experiments on a music database having 2653 pieces of HMM derived from 277 musical pieces. The results are encouraging. The indexing technique pruned out most of the dissimilar songs, and the system used only one seventh of the time for an average music query.

The rest of the paper is organized as follows. We describe the background of music retrieval and hidden Markov model in Section 2. In Section 3 we provide the representation of music by hidden Markov model. The design of the index structure is given in section 4. We describe our experiment design and report the results in Section 5 and we conclude in Section 6.

2. BACKGROUND

The first step in music retrieval is to provide a suitable model to represent each piece of music. A suitable model should not only be loyal to original data but also facilitate retrieval. How to model a song in a music database to facilitate retrieval? Most techniques proposed in the literature model a song as a string, and then, apply traditional text retrieval techniques [3] [2] [6] [5].

Ghias et al. [3] used only three letters of alphabet to characterize the relationship between consecutive pitches: 'U', 'D' and 'S', representing the situation where a note is above, below or the same as the previous notes. Each of the 183 songs in the database and queries in their experiment were converted into a string composed of the three letters. To search the most similar song, they employed approximate string matching proposed in [19], which allowed for note transposition, deletion, and insertion. Search string can be matched with any portion of the melody, rather than just the beginning. McNab et al. [1] [2] also employed an approximate string matching technique. They included tempo, duration, as an matching option for advanced queries. As approximate string matching is a standard application of dynamic programming, the time taken is a matter for some concern.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.

©2002 IRCAM – Centre Pompidou

The approximate matching on 9400 songs took 21 seconds in their test. A faster state matching algorithm [13] was implemented, but it does not discriminate as well as dynamic programming due to the different distance measure of the algorithm. Tseng [7] [6] proposed a pitch profile encoding for queries and an n-note indexing method for approximate matching in sub-linear time. The shorter n-note will match the input query in the presence of random errors, while the longer n-note will favor the ones with more accurate melodies. Downie et al. [4] [5] proposed to use only interval information of the songs to facilitate retrieval. The songs were converted to an interval representation of monophonic melodies and then fragmented into length-n subsections called n-grams. Using n-grams as “musical words”, the authors constructed music database using the text-based, SMART information retrieval system. They analyzed the length of n-gram and the degree of accuracy in representing the intervals by simulated queries using normalized precision and normalized recall.

To characterize the error of hummed notes, researchers are now trying to model these in a probabilistic manner. Probabilistic models for similarity search is first expounded by Ponte et al. [20] and then applied to music by Pickens [21]. A Hidden Markov Model (HMM) describes a doubly stochastic process in which only the current state affects the choice of the next state. They are statistical models for sequential data that have been used successfully in many machine learning applications, especially for speech recognition [11] [10]. Because of its potential ability to capture the commonness of training set and ward off noise, it has been tried in music retrieval recently [12] [16]. In [12], the techniques are adapted from automatic speech recognition to create a melody spotting system in the musical domain. The authors also pointed out that other than dynamic programming for melodic MIR, other comparison techniques which had been successfully applied to other forms of audio signal understanding, particularly those tools for speech recognition, should also be explored in MIR research. Shifrin et al. [16] models the themes of songs as HMMs, and apply a forward algorithm to match a query and HMMs. Their technique is shown to be very effective if the hummed notes have good quality.

3. REPRESENTATION OF THEMES BY HIDDEN MARKOV MODELS

In this section, we describe how to represent a theme by a Hidden Markov Model.

In a *Markov Chain*, we have a finite collection S of *states*. At each time step, the system either stays in the same state or changes to a different state. The probability of the system at a time changing from one state to another state, or staying in the same state, can be represented by a transition probability matrix P . The probability of going from one state to another state does not depend on time, nor on where the system is previous to the current state. This is known as the *Markov Property*.

The directed graph in Figure 1 represents a Markov model of a scalar passage of a theme. States are note transitions, each of which is represented by an interval and an IOIratio, $\langle \text{interval}, \text{IOIratio} \rangle$. Interval is the difference in pitch between two consecutive notes. The *inter onset interval* (IOI_n) is the difference between the onset of notes n and $n+1$, which is the duration of note n , and *IOIratio* is the ratio between

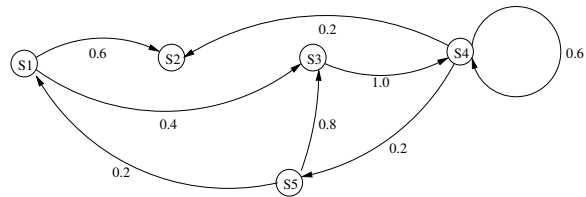


Figure 1: A Markov Model

two consecutive *IOI*. In Figure 1, the numerical value below each directed edge indicates transition probabilities. Only transitions with non-zero probabilities are shown in the figure.

While each musician sings a song slightly differently, casual singers make the difference even more obvious. We assume that the Markov chain created directly from a music theme (a standard theme) is a Hidden Markov Model (*HMM*), and the states extracted from a hummed notes are observable states. The probability distribution of the observable states over the hidden states is based on two heuristic tables and built on the fly [16]. Interval is represented by an integer as there are 12 semi-tones within an octave. IOIratio is recorded as an float value.

4. AN INDEX MECHANISM BASED ON R*-TREE

As both the HMMs and the queries are characterized by state transitions, it is intuitive that the more the transitions of a queries match the transitions of an HMM, the higher probability that this HMM generates the query. However, as observable states in a query include the error made by a singer, a hidden state in an HMM cannot directly match an observable state. Before making the match, hidden states should be converted into a format that includes the error. One could either fold this error into the data representation or into the query. We choose the first, for ease of coding. The latter may be preferred if we expect different query classes with different error probabilities.

4.1 The Representation of a Transition in HMMs

McNab et al. [2] noted there were four types of errors on interval in a hummed notes: *Expansion*, *Compression*, *Repetition* and *Omission*. Expansion error occurs when the subjects sing the small intervals that fall within -4 to 4 . Subjects tends to expand these intervals slightly. Compression error occurs when the subject sing the large intervals, larger than 4 or less than -4 . Subjects tend to compress the intervals slightly. Repetition error occurs when subjects incorrectly repeat notes. Omission error occurs when subjects omit a note. It is difficult to identify repetition and omission errors from the hummed notes. To capture the expansion error and compression errors, a hidden interval is represented by an interval range. In our experiments, we observed that IOIratio also tends to be compressed or expanded by subjects. That is, singers make errors not only in pitch but also in duration. To capture this error, a hidden IOIratio is represented by an IOIratio range. A hidden state is so expanded to a format as

```
<< max_interval, min_interval >,
    < max_IOIratio, min_IOIratio >>
```

where `max_interval` and `min_interval` are the maximum and minimum intervals that are commonly observable interval deviated from accurate interval, the hidden interval, so are

max_IOIratio and min_IOIratio. By expanding a hidden state to two ranges, a transition in an HMM can be expanded to four ranges:

```
start_ << max_interval, min_interval >,
        < max_IOIratio, min_IOIratio >>
end_ << max_interval, min_interval >,
        < max_IOIratio, min_IOIratio >>
```

These four ranges can be represented by a four dimensional box.

4.2 The Representation of a Transition in a Query

As an observable state is depicted by a duple, <interval, IOIratio>, a transition in a query, an observable transition, is represented by

```
start_ < interval, IOIratio >,
end_ < interval, IOIratio >
```

which can be represented by a point in a four dimensional space.

4.3 Create a Tree and Search in the Tree

If a point of four dimension is in a four dimensional box, the corresponding observable note transition can be represented by the hidden transition with high probability. We say the observable transition matches the hidden transition.

To index multi-dimensional data, the most popular index technologies in databases are from the R-tree [14] family. Here we use R*-tree [15] because of its high efficiency for searching. All the transition boxes of each HMM are inserted into a single R*-tree of four dimensions. The following is the Create_Tree Algorithm to create the index.

```
Create_Tree
repeat
  read in an HMM
  repeat
    get a hidden transition
    change it into a four-dimensional box
    insert the box into R*-tree
  until no more hidden transition in the HMM
until no more HMM
```

When a query is submitted, it is changed to a sequence of state transitions and each transition is represented by a four dimensional point. Thus each query is represented as a set of points. These points are used to search the R*-tree. The number of each HMM's boxes containing the points are counted. The returned result is the ranked HMMs, which have the high probability to be the expected HMMs. The following is the Search_Rank Algorithm.

```
Search_Rank
read in the query and change it into a string of
  query points
repeat
  get one point
  submit the point to the R*-tree
  record the HMMs that has as least a box
    containing this point
until no more points in the query
count the number of matches of each HMM
rank these HMMs
```

After the ranked HMMs are returned, the forward algorithm is employed to do post-processing to retrieve the most similar

ones. As most HMMs are pruned out in the Search_Rank, the forward algorithm processes fewer HMMs and the total cost is greatly reduced.

5. EXPERIMENT

In this section, we describe our experiment design and test results.

5.1 Experiment setup

In our test database, there were a corpus of 277 songs (used in [16]). Themes, the musical equivalent of keywords, were extracted from each piece of music by MME [18]. An average of 9.58 themes per piece of song were found by MME, resulting in a database of 2653 monophonic themes. HMMs of the themes were then generated automatically and placed in the database.

We also used the query corpus used in [16]. 4 subjects sung in 6 pieces of music each, totally 24 queries. Using the forward algorithm, 10 of the queries were returned with the expected song at the top of the ranked results. We denote these queries as "good" queries. For the other queries, the top song returned was not the one expected. For most of these failures, it was because that the hummed notes were of bad quality, far from the expected one. Some failures were caused by the poorly extracted notes. The pitch tracker skipped so many notes that the extracted notes were difficult to recognize when played as MIDI. Among these 14 queries, there were 4 queries which the expected song was among the top 10 returned songs searched by the forward algorithm. We denote these 4 queries as "decent". The other 10 queries, which the expected song was not in the top 10 returned, are denoted as "bad". In our experiments, we used the forward algorithm to do post processing after the candidate HMMs were selected by the R*-tree.

As the forward algorithm was employed to do post-processing, "bad" queries' expected answer can not be returned, we used only the 14 "good" and "decent" queries as our test queries,

It is intuitive that if the ranges of the boxes are set to be large, large number of candidate HMMs would be returned by R*-tree. Much time should be spent in post processing. In contrast, if the ranges of the boxes are set to be small, few candidate HMMs would be selected by R*-tree. This would make the system fast. However, too small boxes that tolerate little error might prune expected HMMs out. So, it is important to choose a good balance.

Since interval varies by the step of 1, we set

$$Max_interval = hidden_interval + \alpha$$

and

$$Min_interval = hidden_interval - \alpha$$

As IOIratio is the ratio of consecutive durations, we set the range of IOIratio by

$$Max_IOIratio = hidden_IOIratio * \beta$$

and

$$Min_IOIratio = hidden_IOIratio / \beta$$

In our experiments, we set α and β with different values to find the suitable ones.

Our test was run on a Microsoft Windows 2000, Pentium III machine.

5.2 Experiment Results and Sensitivity Analysis

In the first set of tests, we set $\alpha = 0$ and $\beta = 1.1, 1.2$ and 1.3 . Although the processing time using the index was much less than the time using only the forward algorithm, several queries did not get the expected answer. The average response time and the number of queries missing expected HMM in different settings are given in Table 1. The reason

Table 1: Response time and missing queries with different α and β

$\alpha = 0$	Forward Algorithm	$\beta = 1.1$	$\beta = 1.2$	$\beta = 1.3$
Average response time (s)	2.73	0.28	0.35	0.39
Number of queries missing expected HMM	0	3	1	1

for this poor performance is that α was set to be 0, expecting there is no error in interval. So, for those queries that were not of high quality, the expected HMMs were pruned out by index.

In the second set of tests, α was set to be 1 and $\beta = 1.1, 1.2$ and 1.3 . All the 14 queries were returned with the expected HMM. The average response time is listed in Table 2. The

Table 2: Average response time with different α and β

$\alpha = 1$	Forward Algorithm	$\beta = 1.1$	$\beta = 1.2$	$\beta = 1.3$
Average response time (s)	2.73	0.49	0.67	0.77

best case is that when $\alpha = 1$ and $\beta = 1.1$, the average response time is 0.49 seconds while the average response time is 2.73 seconds by the forward algorithm. The average response time increases with the increasing of β . With larger β , more candidate HMMs would be returned by R*-tree and the time used in post processing was increased.

To take the common errors made by the subject into consideration, as described in section 4.1, we set α differently and did another set of test. When $hidden_interval > 4$, $Max_interval = hidden_interval$ and $Min_interval = hidden_interval - 1$. If $0 < hidden_interval < 4$, $Max_interval = hidden_interval + 1$ and $Min_interval = hidden_interval$. When $hidden_interval = 0$ or 4 , set $\alpha = 1$. Symmetrically with the negative intervals. All the 14 queries were returned with the expected HMM. The results are reported in Table 3. Compared to the previous test, the average response time is

Table 3: Average response time with setting α as described

set α as described	Forward Algorithm	$\beta = 1.1$	$\beta = 1.2$	$\beta = 1.3$
Average response time (s)	2.73	0.36	0.46	0.52

even shorter. Because this settle narrow down the interval range, the number of candidate HMMs returned by R*-tree was reduced, so was the time of post processing.

For the setting of $\beta = 1.1$, each individual queries' response time is displayed in Figure 2. From Figure 2, it can be seen,

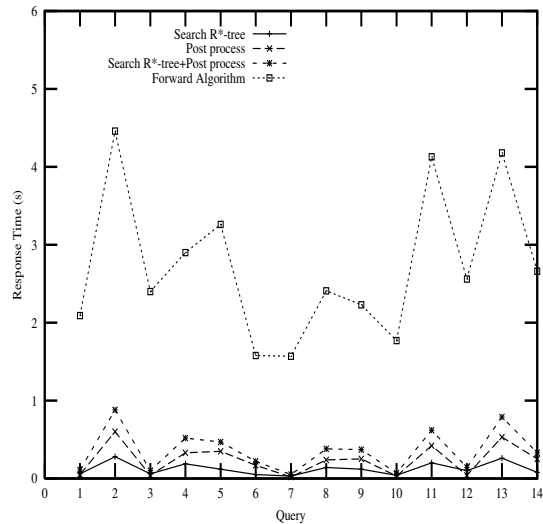


Figure 2: The response time of each individual query

for the “good” and “decent” queries, all the correct answers were returned with much shorter response time where the index was used. Different queries cost different time. It is because different queries had different number of notes and transitions. The larger the number of notes in a query, the longer time it takes to search the R*-tree and carry out the forward algorithm. The average response time using the index mechanism is 0.36 seconds per query, while the average response time using only the forward algorithm is 2.73 seconds per query. We achieved average 7 fold speedup using the index mechanism.

For the “bad” queries, we could not get the expected songs in the candidate HMMs returned by the index. It is because of the low quality of the queries. The error in the queries is so large that the point representing a transition is not located in a transition box expected. So, these queries find some songs other than the ones supposed.

6. CONCLUSION

In this work, we proposed a novel idea to use short ranges to tolerate errors in hummed notes. Each transition in HMMs is transformed into a four-dimensional box and indexed in a R*-tree. Transforming queries into four-dimensional points, candidate songs with high probability can be selected out quickly. By assigning the parameter of the ranges properly, we achieved a seven-fold speed up using this index mechanism than the original forward algorithm in our experiment. For larger datasets, we expect to be able to show a large speed up.

7. ACKNOWLEDGMENT

We gratefully acknowledge the support of the National Science Foundation under grand IIS-0085945, and The University of Michigan College of Engineering seed grand 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 wish to thank Shifrin for providing us the test data. We are also grateful for insightful discussions with Professor William Birmingham and his students.

8. REFERENCES

- [1] R.J. McNab, L.A. Smith, I.H. Witten, C.L. Henderson and S.J. Cunningham. Towards the Digital Music Library: Tune Retrieval from Acoustic Input. In *Digital Libraries '96: Proceedings of the ACM DL Conference*.
- [2] R.J. McNab, L.A. Smith, D. Bainbridge and I.H. Witten. The New Zealand Digital Library MELody inDEX. In *D-Lib Magazine*, May 1997.
- [3] A. Ghias, J. Logan, D. Chamberlin and B.C. Smith. Query By Humming: Musical Information Retrieval in An Audio Database. In *ACM Multimedia*, 1995.
- [4] J.S. Downie. Music Retrieval as Text Retrieval: Simple Yet Effective. In *SIGIR*, 1999.
- [5] J.S. Downie and M. Nelson. Evaluation of a Simple and Effective Music Information Retrieval Method. In *SIGIR*, 2000.
- [6] Yuen-Hsien Tseng Content-Based Retrieval for Music Collections. In *SIGIR*, 1999.
- [7] Yuen-Hsien Tseng Crystal: A Content-based Music Retrieval System. In *SIGIR*, 1999.
- [8] A. Uitdenbogerd and J. Zobel. Melodic Matching Techniques for Large Music Databases. In *ACM Multimedia*, 1999.
- [9] Jyh-Shing Ronger Jang, Hong-Ru Lee and Chia-Hui Yeh. Query by Tapping: A New Paradigm for Content-Based Music Retrieval from Acoustic Input. In *IEEE Pacific Rim Conference on Multimedia*, 2001.
- [10] Y. Bengio. Markovian Models for Sequential Data. In *Statistical Science*, 97.
- [11] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proceedings of the IEEE*, 1989.
- [12] A.S. Durey and M.A. Clements. Melody Spotting Using Hidden Markov Models. In *MIR*, 2001
- [13] S. Wu and U. Manber. Fast Text Searching Allowing Errors. In *Communication of ACM*, Vol. 10, 2001
- [14] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD Conference*, 1984.
- [15] N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM SIGMOD Conference*, 1990.
- [16] J. Shifrin, B. Pardo, C. Meek and W. Birmingham. HMM-Based Musical Query Retrieval. In *Joint Conference on Digital Libraries 2002*.
- [17] W.P. Birmingham, R.D. Dannenberg, G.H. Wakefield, M. Bartsch, D. Bykowski, D. Mazzoni, C. Meek, M. Melody, W. Rand. MusArt: Music Retrieval Via Aural Queries. In *Proceeding of ISMIR2001*.
- [18] C. Meek and W.P. Birmingham. Thematic Extractor. In *Proceeding of ISMIR2001*.
- [19] R.A. Baeza-Yates and C.H. Perleberg. Fast and Practical Approximate String Matching. In *Combinatorial Pattern Matching, Third Annual Symposium*, 1992.
- [20] J.M. Ponte and W.B. Croft. A Language Modeling Approach to Information Retrieval. In *SIGIR*, 1998.
- [21] J. Pickens. A Comparison of Language Modeling and Probabilistic Text Information Retrieval Approaches to Monophonic Music Retrieval. In *Proceeding of ISMIR2000*.