QUERY BY HUMMING OF MIDI AND AUDIO USING LOCALITY SENSITIVE HASHING

Matti Ryynänen and Anssi Klapuri

Tampere University of Technology
Institute of Signal Processing
P.O.Box 553, FI-33101 Tampere, Finland
{matti.ryynanen, anssi.klapuri}@tut.fi

ABSTRACT

This paper proposes a query by humming method based on locality sensitive hashing (LSH). The method constructs an index of melodic fragments by extracting pitch vectors from a database of melodies. In retrieval, the method automatically transcribes a sung query into notes and then extracts pitch vectors similarly to the index construction. For each query pitch vector, the method searches for similar melodic fragments in the database to obtain a list of candidate melodies. This is performed efficiently by using LSH. The candidate melodies are ranked by their distance to the entire query and returned to the user. In our experiments, the method achieved mean reciprocal rank of 0.885 for 2797 queries when searching from a database of 6030 MIDI melodies. To retrieve audio signals, we apply an automatic melody transcription method to construct the melody database directly from music recordings and report the corresponding retrieval results.

***Index Terms*—** Music, Information retrieval, Database query processing, Audio Systems

1. INTRODUCTION

Query by humming (QBH) refers to music information retrieval systems where short audio clips of singing or humming act as queries. In a normal use case of QBH, a user wants to find a song from a large database of music recordings. If the user does not remember the name of the artist or the song to make a metadata query, a natural option is to sing, hum, or whistle a part of the melody of the song into a microphone and let a QBH system to retrieve the song.

The QBH task can be broadly divided into two subproblems: i) converting a query into a format which enables robust searching and ii) matching the query with melodies in the database. The former problem is often associated with automatic transcription of a query into temporally segmented note events or into frame-wise measured pitch trajectory, whereas the latter concentrates on measuring melodic similarity. See [1] for an overview of music information retrieval systems and research on melodic similarity.

There exist lots of QBH studies in the literature. Most approaches use either note or pitch sequence to represent the query. Matching approaches include string matching techniques [2], hidden Markov models [3, 4], and dynamic programming [5]. A number of the current state-of-the-art QBH systems have been evaluated in Music Information Retrieval Evaluation eXchange (MIREX) 2006 and 2007.

The major challenges for QBH systems include i) handling of highly varying quality of queries, ii) huge size of melody databases, and iii) automatic production of the melody databases. First, the quality of queries may vary drastically in terms of staying in tune and tempo and also in the recording quality of the query audio. Second, linear search over database items is not acceptable due to huge databases of music. Third, most of the QBH research has concentrated on searching from databases of MIDI melodies and it would be highly desirable to obtain such databases directly from music recordings. From an application point of view, this would also enable immediate playback of the retrieved melody segments in the original music piece.

This work was supported by the Academy of Finland, project No. 213462 (Finnish Centre of Excellence program 2006–2011).

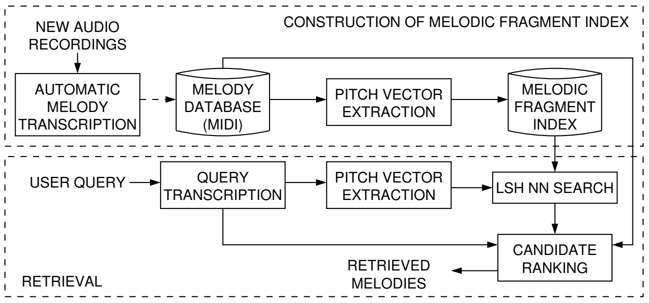

Fig. 1. A block diagram of the proposed method.

We propose a robust QBH method with sublinear search time over database items. Figure 1 shows a block diagram of the method. Given a database of melodies in MIDI format, the method constructs an index of melodic fragments by extracting pitch vectors. A pitch vector stores an approximate representation of melody contour within a fixed-length time window. In retrieval, the method automatically converts a query into MIDI notes and then extracts pitch vectors. For each query pitch vector, the method searches for nearest neighbors in Euclidean space from the index of database melody fragments to obtain melody candidates and their matching positions in time. This can be performed very efficiently by using locality sensitive hashing (LSH). Final ranking of candidates is done by comparing the whole transcribed query to each candidate melody segment. Due to the melodic fragment index, the method manages long database melodies directly, without having to segment melodies into phrases. Also, the queries do not have to start from the beginning of a melodic phrase. Using LSH provides a significant speed-up and retrieval performance comparable to the state-of-the-art.

To retrieve audio signals, we demonstrate the use of an automatic melody transcription method to produce melody database directly from music recordings and achieve very encouraging results. Experimental results for QBH of audio were reported only very recently [6].

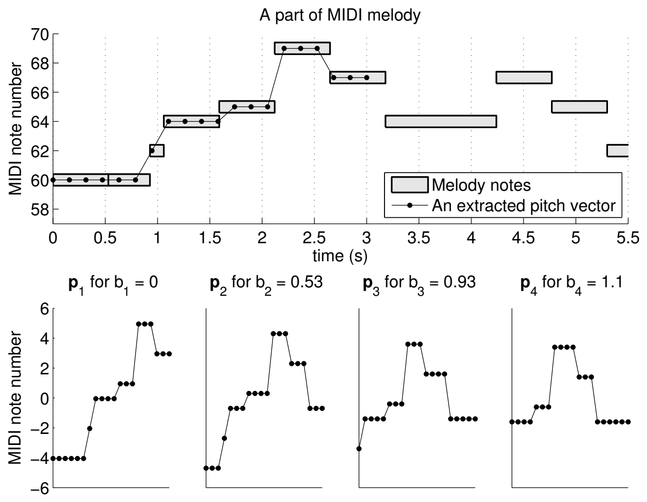

Fig. 2. An example of pitch vector extraction.

2. CONSTRUCTING INDEX OF MELODIC FRAGMENTS

The term melodic fragment refers here to a melody pitch contour within a fixed-length time window. The method constructs an index which stores melodic fragments, their temporal positions within the database melodies, and melody identifiers. The melody identifier determines the song from which a melody fragment has been extracted. The index enables efficient retrieval of melodies from the database.

2.1. Pitch Vector Extraction

A melody is here defined as a sequence of L notes $n_{1:L}$, where i:th note $n_i = \langle p_i, b_i, e_i \rangle$ is defined by pitch p_i in MIDI note numbers, and the onset time b_i and the offset time e_i of the note in seconds. The melodic fragments are represented as pitch vectors which are extracted from such note sequences.

Given a melody $n_{1:L}$, the pitch vectors are extracted as follows. First, rests between consecutive notes are removed by extending the offset of a preceding note to the onset of the following note, i.e., $e_i \leftarrow b_{i+1}$ for $i = 1, \ldots, L-1$. Then for each note i, a pitch vector $\mathbf{p}_i$ of length d is extracted by determining the melody pitch values within w-second window starting from the note onset b_i. More exactly, the pitch values are determined on a uniform time grid $b_i + j\frac{w}{(d-1)}, j = 0, \ldots, d-1$. No pitch vector is extracted if the window exceeds the offset of the last note, i.e., $b_i + w > e_L$. The resulting pitch vector $\mathbf{p}_i$ is then normalized to have zero mean. All-zero pitch vectors are ignored due to their low information content. Figure 2 shows an example of pitch vector extraction by using window size $w = 3$ s and vector length $d = 20$. The top panel shows a part of a melody and the bottom panels show the first four pitch vectors $\mathbf{p}_i$ and their extraction positions b_i. The above-mentioned parameter values worked best in our experiments.

The melodic fragment index is constructed by extracting the pitch vectors from each database melody. The resulting index contains a list of $\langle s, b_i, \mathbf{p}_i \rangle$ records, where s identifies the song, b_i defines the extraction position within the song, and $\mathbf{p}_i$ represents the melodic fragment.

2.2. Similarity of Melodic Fragments

The similarity of melodic fragments is here measured using Euclidean distance between pitch vectors. Formally, two pitch vectors $\mathbf{p}_i$ and $\mathbf{p}_j$ define points in d-dimensional space where the distance is given by

$$\|\mathbf{p}_i - \mathbf{p}_j\| = \left(\sum_{k=1}^{d} |\mathbf{p}_i(k) - \mathbf{p}_j(k)|^2 \right)^{(1/2)} . \quad (1)$$

Euclidean distance is not only simple but also appears to be very effective measure for similarity (see, e.g., [7]). Given a melodic fragment defined by point $\mathbf{p}_i$, we can find similar fragments in the index by searching for nearest neighbors (NNs) of the point, i.e., all the points to which the distance is less than a specified threshold r. This could be done by simply measuring the distance of $\mathbf{p}_i$ to all the vectors in the database. However, this results in a search time that depends linearly on the database size.

To obtain a sublinear time complexity, we use locality sensitive hashing [8, 9]. LSH is a randomized algorithm for searching approximately nearest neighbors in high dimension spaces. The idea is that the points whose distances are within the threshold r will be hashed to a same bucket with a certain probability. This user-defined probability controls the trade-off between the accuracy and the speed of LSH. In our method, we employ LSH implementation package E2LSH by the original authors (see http://web.mit.edu/andoni/www/LSH). Based on the implementation, we built a server for handling the melodic fragment index and a client for retrieving similar melodic fragments from the database. Recently, LSH has been applied, e.g., in remixed music recognition [10] and in audio fingerprinting [11].

3. QUERY PROCESSING AND MELODY RETRIEVAL

The retrieval stage consists of the following steps: i) transcription of a sung query into notes, ii) extraction of pitch vectors from the notes, iii) retrieving similar melodic fragments in the database using LSH, and iv) performing final ranking of the retrieved melody candidates. The following subsections explain these steps in more detail.

3.1. Query Transcription and Tuning

A sung query is first converted into a note sequence. For this task, we use a melody transcription method designed for polyphonic music. Although it is not necessary to handle polyphony in query transcription, this method is used also to produce a melody database directly from music recordings (see Fig. 1). The method is an improved version of [12]. Briefly, the method uses a frame-wise pitch salience estimator to measure the strength of different fundamental frequencies in 92.9 ms analysis frames with 23.2 ms interval between successive frames. This feature extractor is followed by HMMs representing melody notes and the background. The method also applies a musicological model to control between-note transitions. As an output, the method produces a sequence of notes in the format introduced in Sec. 2. One could also use some other melody transcription method which produces note sequences, e.g., the method proposed in [12].

The queries are not likely performed in absolute tuning (MIDI note 69 at 440 Hz), i.e., the tuning of a sung note can be between two integer MIDI pitches. Therefore, each transcribed query note is tuned by shifting it in frequency at most half a semitone up or down so as to maximize pitch salience within the note. Figure 3 shows an example of transcribed and tuned query notes with the estimated pitch saliences.

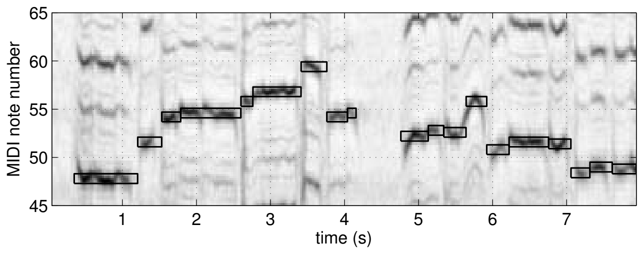

Fig. 3. A transcribed query. The grey-level intensity indicates the estimated pitch salience and the black boxes show the transcribed and tuned query notes.

3.2. Retrieval of Candidate Melodies

The tuned query note sequence is then used to retrieve similar melodic fragments from the database by extracting pitch vectors from the query notes as explained in Sect. 2.1. It is important to notice that a user may sing a song in a different tempo compared to the stored melodic fragments in the index. Therefore, the method extracts several pitch vectors with different window sizes for each note. Let m_j denote j:th window size modifier. For each query note onset b_q, the method samples M pitch vectors using window sizes wm_j, $j = 1, \ldots, M$. A modifier value $m < 1$ implies that a query melodic fragment is performed faster than the database melodic fragment. Respectively, a slower performance is indicated by modifier values greater than one. A reasonable range for window modifier values is between 0.65–1.7.

For each query pitch vector, the method then searches for similar melodic fragments in the database using LSH. The LSH returns the nearest neighbors and their distances to the query point as matches. Our simulations indicated that it is sufficient to preserve only a few smallest-distance matches per each query point. After retrieving matches for all the query points, we have a list of candidate melodies for final ranking.

3.3. Final Ranking

To obtain the final list of retrieved melodies, the candidate melodies are ranked according to their distance to the entire query note sequence. The ranking is performed by examining all the matches preserved in the previous step. One match is denoted by $\langle b_q, m_j, b_c, s\rangle$, where b_q is the extraction position of the query pitch vector, m_j is the used window size modifier, b_c is the extraction position of the database melodic fragment, and s is the song identifier. In addition, let $t_0^{(q)}$ and $t_1^{(q)}$ denote the onset time of the first query note and the offset time of the last query note, respectively. Then the time region corresponding to the entire query in the candidate melody is defined by $t_0^{(c)} = b_c - (b_q - t_0^{(q)})/m_j$ and $t_1^{(c)} = b_c + (t_1^{(q)} - b_q)/m_j$. Hereafter, the term candidate segment refers to this time region within the candidate melody. The upper panel in Fig. 4 shows an example how the candidate segment is determined for one match.

The entire query and the candidate segment are then normalized both in pitch and time for distance calculation. A note sequence $n_{1:L}$ is normalized by performing the operations $p_i \leftarrow p_i - \bar{p}$, $b_i \leftarrow (b_i - b_1)/t_{\text{dur}}$, and $e_i \leftarrow (e_i - b_1)/t_{\text{dur}}$ for $j = 1, \ldots, L$ where $\bar{p} = \left(\sum_{i=1}^{L} p_i(e_i - b_i)\right)/t_{\text{dur}}$ is the mean pitch and $t_{\text{dur}} = e_L - b_1$ is the duration of the sequence. The lower panel in Fig. 4 shows the normalized query and the normalized candidate segment.

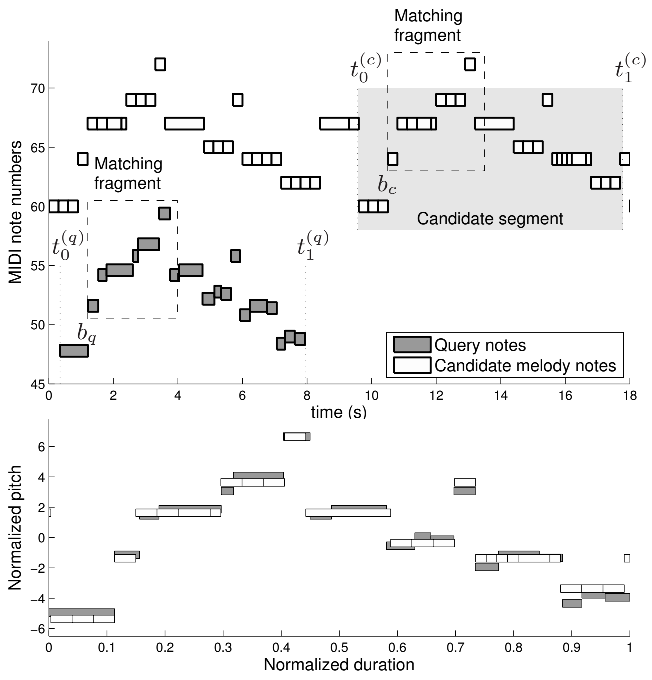

Fig. 4. A query and a matching candidate melody. The upper panel shows the matching fragments in these melodies and the determined candidate segment (light grey area). The lower panel shows the normalized query and the normalized matching segment for distance evaluation. See text for details.

The distance between the normalized query and the normalized candidate segment is evaluated by using recursive alignment (RA) proposed by Wu et al. [7]. Briefly, RA uses top-down approach to temporally align two note sequences to minimize their distance in frequency. First, the method divides the candidate segment into two halves at position 0.5. The query is also divided to two halves but using several possible division points, e.g., 0.45, 0.5, and 0.55. For each division point, the two halves are compared to the halves in the candidate segment to measure the distance. The division point which gives the smallest joint distance for the halves is preserved. Then the method applies the above procedure for the decided halves independently in a recursive manner. In other words, RA finds the best global alignment first and then recursively continues to align the smaller segments. As a result, RA returns the distance between the aligned note sequences. For details, see [7].

The above distance evaluation is performed for each match. The minimum distance match per candidate melody is preserved, since there may exist several candidate segments per melody. Finally, the list of candidates is sorted in ascending distance order and returned to the user. If melodies are retrieved from audio, also the candidate segments are returned to enable playback of the retrieved melodies in the original music recordings.

4. RESULTS

The method performance is measured using mean reciprocal rank (MRR) and top-X hit rate criteria. For the i:th query, let r_i denote the rank of correct answer in the retrieved melodies. MRR is then

Table 1. Melody retrieval results.

			MRR	Top-X hit rate (%)				
Corpus	N	D_{sz}		1	3	5	10	20
Jang	2797	6030	0.885	86	90	91	92	93
	2797	2048	0.909	89	92	93	94	95
Music	159	427	0.578	52	58	62	69	73

given by $(1/N)\sum_{i=1}^{N} r_i^{-1}$ where N is the number of queries. The top-X hit rate reports the proportion of queries for which $r_i \leq X$.

First, we evaluated the method using Roger Jang's corpus consisting of $N = 2797$ eight-second queries and 48 ground-truth MIDI files (available at http://www.cs.nthu.edu.tw/∼jang). We add 5982 melodies from the Essen Associative Code and Folksong database (EsAC) to obtain a database of $D_{\text{sz}} = 6030$ melodies. Secondly, we have a set of 159 queries of 32 different pop songs. In 97 queries, the users performed the melodies as they remembered them. The rest of the queries were performed while listening to the piece. We used the melody transcription method to produce a melody database directly from 427 full music recordings, including the 32 pop songs. Table 1 summarizes our results on both databases.

For the Jang's corpus, the method reached MRR of 0.885 and top-3 hit rate of 90%. For these results, it was sufficient to preserve only two smallest-distance matches per query point (see Sect. 3.2) which returned on the average 134 candidate melodies for a query. Interestingly, MRR of 0.592 could be reached just by ranking the candidate melodies according to the number of matching melodic fragments. The used number M of window size modifiers m_j was 17, and recursive alignment was used with five possible division points and two recursion levels. With a Matlab implementation running on a 3.2 GHz Pentium 4 processor, the mean query time was 4.5 seconds of which the query transcription takes approximately 30%, LSH 32%, and the final ranking 13%. LSH provided a speed-up of factor 4–20 compared to exact nearest neighbor search in candidate retrieval without losing any accuracy in results. Retrieval errors are mostly related to the query performances: some queries differ from the correct answer so much that matching is very challenging even for human.

For a suggestive comparison, the method submitted by Wu and Li (essentially the method published in [7]) performed best in the MIREX 2006 evaluation with MRR of 0.900 for the Jang's corpus. In that evaluation, the corpus was extended with 2000 EsAC melodies (i.e., $D_{\text{sz}} = 2048$) and the method was required to search for matches anywhere in the database melodies similar to our simulations (see www.music-ir.org/mirex2006/index.php/QBSH:_Query-by-Singing/Humming_Results for results and abstracts). We evaluated our method also for 2048 melody database and the results are given on the second row in Table 1. However, our results are not exactly comparable to MIREX 2006 evaluation since the included EsAC melodies are not exactly the same ones in these experiments.

The results for the automatically transcribed melody database are expectedly worse than with the manually prepared MIDI files in Jang's corpus. However, the method achieved MRR of 0.578 which is rather encouraging result and motivates for further study. This result compares favorably to previously reported MRRs for a 200-piece database [6]. For some audio recordings, the transcribed melodies still contain too many note insertions or deletions in order to perform successful retrieval. Audio demonstrations of retrieval from music recordings are available at http://www.cs.tut.fi/sgn/arg/matti/demos/qbh.

5. CONCLUSIONS

We have proposed a QBH method based on LSH and achieved retrieval performance comparable to the state-of-the-art. The method also achieved promising results for QBH of audio. The current representation of melodic fragments enables accurate results but contains redundant information and consequently uses memory inefficiently. Future work includes development of a more compact representation. In addition, retrieval directly from music seems very promising for future development.

6. ACKNOWLEDGMENTS

The authors would like to thank Roger Jang for providing a fine corpus for QBH research and Alexandr Andoni for the LSH implementation and advice.

7. REFERENCES

[1] R. Typke, *Music Retrieval based on Melodic Similarity*, Ph.D. thesis, Universiteit Utrecht, 2007.

[2] K. Lemström, *String Matching Techniques for Music Retrieval*, Ph.D. thesis, University of Helsinki, 2000.

[3] C. Meek and W. Birmingham, "Applications of binary classification and adaptive boosting to the query-by-humming problem," in *Proc. 3rd International Conference on Music Information Retrieval*, 2002.

[4] J.-S. R. Jang, C.-L. Hsu, and H.-R. Lee, "Continuous HMM and its enhancement for singing/humming query retrieval," in *Proc. 6th International Conference on Music Information Retrieval*, 2005.

[5] J.-S. R. Jang and M.-Y. Gao, "A query-by-singing system based on dynamic programming," in *Proc. International Workshop on Intelligent Systems Resolutions*, 2000.

[6] A. Duda, A. Nürnberger, and S. Stober, "Towards query by humming/singing on audio databases," in *Proc. 7th International Conference on Music Information Retrieval*, 2007.

[7] X. Wu, M. Li, J. Yang, and Y. Yan, "A top-down approach to melody match in pitch countour for query by humming," in *Proc. International Conference of Chinese Spoken Language Processing*, 2006.

[8] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, 2006, pp. 459–468.

[9] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. ACM Symposium on Computational Geometry*, 2004, pp. 253–262.

[10] M. Casey and M. Slaney, "Fast recognition of remixed music audio," in *Proc. 2007 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2007.

[11] M. Covell and S. Baluja, "Known-audio detection using Waveprint: Spectrogram fingerprinting by wavelet hashing," in *Proc. 2007 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2007.

[12] M. Ryynänen and A. Klapuri, "Transcription of the singing melody in polyphonic music," in *Proc. 7th International Conference on Music Information Retrieval*, 2006.