Scalable Content-Based Music Retrieval Using Chord Progression Histogram and Tree-Structure LSH

Yi Yu, Roger Zimmermann, *Senior Member, IEEE*, Ye Wang, *Member, IEEE*, and Vincent Oria

***Abstract*—With more and more multimedia content made available on the Internet, music information retrieval is becoming a critical but challenging research topic, especially for real-time online search of similar songs from websites. In this paper we study how to quickly and reliably retrieve relevant songs from a large-scale dataset of music audio tracks according to melody similarity. Our contributions are two-fold: (i) Compact and accurate representation of audio tracks by exploiting music semantics. Chord progressions are recognized from audio signals based on trained music rules, and the recognition accuracy is improved by multi-probing. A concise chord progression histogram (CPH) is computed from each audio track as a mid-level feature, which retains the discriminative capability in describing audio content. (ii) Efficient organization of audio tracks according to their CPHs by using only one locality sensitive hash table with a tree-structure. A set of dominant chord progressions of each song is used as the hash key. Average degradation of ranks is further defined to estimate the similarity of two songs in terms of their dominant chord progressions, and used to control the number of probing in the retrieval stage. Experimental results on a large dataset with 74,055 music audio tracks confirm the scalability of the proposed retrieval algorithm. Compared to state-of-the-art methods, our algorithm improves the accuracy of summarization and indexing, and makes a further step towards the optimal performance determined by an exhaustive sequence comparison.**

***Index Terms*—Audio computing, chord progression histogram, locality sensitive hashing, music-IR, tree-structure.**

I. Introduction

With an explosive growth of community-contributed multimedia data, content-based music information retrieval (CBMIR) on large-scale social websites has become a timely and critical research topic. For example, many music movie soundtracks, with the same or similar melody but sung and recorded by different people, are uploaded to YouTube every year. A melody is a linear succession of music tones. CBMIR, in terms of melody similarity, has several novel applications such as plagiarism analysis, near duplicate audio detection, relevant song retrieval and recommendation, etc. In typical scenarios, a user can find audio tracks similar to his favorite melody using an audio example, or music companies can recommend to users new music albums with similar melodies according to listening records. These applications need large-scale CBMIR techniques.

Scalable CBMIR is commonly related to two essential aspects: (i) Representing audio signals by compact features. Music signals usually are described by sequences of low-level features such as short-time Fourier transform (STFT) [1], pitch [2], [3], mel-frequency cepstral coefficient (MFCC), and chroma [4]–[6]. (ii) Organizing audio features in the database using an indexable format. Locality sensitive hashing (LSH) [1], [5], [7], [8], tree structure [9]–[11], and hierarchical structure [12] are typical methods to music audio content indexing. These two aspects need joint design so as to improve both accuracy and efficiency of large-scale music retrieval. Unfortunately, music audio content analyses and summarizations by means of low-level features in previous works are insufficient for the scalable CBMIR task. This is because low-level feature descriptors of audio signals are highly heterogeneous and do not generate a robust description for music audio tracks. The performance of summarizing audio signals without exploiting music knowledge is limited. In contrast, mid-level features (chord [13]–[18], rhythm, and instrument) represented as musical attributes are able to better extract music structures from complex audio signals and retain semantic similarity. A chord sequence contains rich music information related to tonality and harmony, which is helpful for effectively distinguishing whether music audio tracks are similar to each other or not. However, chord recognition accuracy is still relatively low in previous state-of-the-art algorithms [13]–[18], which affects the performance of chord-based music retrieval. Moreover, good scalability requires that the retrieval complexity should be at most sub-linear with respect to the number of songs in the database. These issues inspire us to make use of mid-level feature descriptors and organize them in an indexable structure affordable for large-scale music audio content matching and retrieval.

In this paper we study chords—a harmony-related mid-level feature—for the task of scalable CBMIR and exploit chord progressions (CPs) to realize accurate summarization of music content and efficient organization of the database. As for CPs, we mainly consider transitions between adjacent chords. But it is easy to extend the idea to longer chord progressions. The proposed algorithm consists of three key components: (i) recognizing CPs from a music audio track based on the trained music rules, (ii) computing a summary of the track from the recognized CPs, and (iii) organizing the summaries of audio tracks in the database using an indexing structure. How to improve CP accuracy was investigated in our previous work [19]. In this paper we conduct more detailed analyses. Specifically, the CPs

Manuscript received September 22, 2012; revised January 06, 2013 and March 14, 2013; accepted March 15, 2013. Date of publication June 18, 2013; date of current version November 13, 2013. This work was supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Chia-Wen Lin.

Y. Yu, R. Zimmermann, and Y. Wang are with the Department of Computer Science, National University of Singapore, Singapore (e-mail: yuy@comp.nus.edu.sg).

V. Oria is with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2013.2269313

are divided into two categories. Their recognition accuracy is investigated by using multiple chroma features and refining the training set. In addition, summaries computed from CPs are organized in a refined LSH table in order to accelerate the retrieval process. The effect of CP recognition on LSH design is also theoretically analyzed. Our major contributions are summarized as follows:

- *Compact and accurate representation of audio tracks via CPs*. Recognition accuracy of CPs is greatly improved by exploiting multi-probing. More specifically, through a modified Viterbi algorithm, N-best CPs are locally probed, which in terms of their ranks are further summarized into a compact *chord progression histogram* (CPH). Combinations of the proposed multi-probing technique with different chroma features and the effect of refining the training set are also studied.
- *Efficient organization of CPHs of audio tracks via LSH*. A satisfactory recall is achieved with only one hash table, by using dominant CPs of audio tracks as hash keys and exploiting multi-probing. Organizing songs in the layered tree-structure helps alleviate the potential imbalance among buckets. Average degradation of ranks is further defined to assess the similarity of two songs in terms of their dominant CPs, and used to control the number of probings in the retrieval stage.

By locally probing CPs among adjacent frames according to their state transition probabilities, the computed CPH is an accurate and powerful feature containing harmonic progression and tonal structures of audio tracks. In addition, retrieval accuracy of the LSH-based indexing is improved by multi-probing, and the implementation of LSH is efficient by requiring only one hash table. Our experiments, on real-world large-scale datasets including 74,055 audio tracks, confirm that the proposed algorithm achieves a nice tradeoff between retrieval accuracy and efficiency and demonstrate the feasibility of using CPs for music content representation and scalable retrieval. Compared to previous schemes which address summarization and indexing, the proposed algorithm makes a further step towards the optimal performance determined by an exhaustive sequence comparison. As melody is usually embodied in chord progressions, the proposed CPH feature serves as a signature of an audio melody and helps to accelerate similarity retrieval in terms of melody.

This work begins with a review of music background and related work on music representations, LSH, CBMIR, and a comparison with our retrieval method in Section II. Section III describes the proposed retrieval algorithm in detail, focusing on how to realize multi-probing in recognizing CPs, how to summarize the probed CPs into a CPH, and how to organize CPHs in the LSH table. In Section IV, we discuss the effect of training sets on CP recognition, the effect of multi-probing in CP recognition and retrieval, and present overall experiment results. Finally, we conclude the paper with Section V.

II. Background and Related Work

Conventionally, music retrieval on the Internet heavily depends on tag information, both in the database and the query. However, tag information of user-generated audio tracks might be missing, ambiguous, or even misleading. In contrast, content analysis and detection of audio tracks help improve retrieval quality. However, scalability becomes a challenging issue as multimedia music content has become prevalent on user-contributed social websites. To provide real-time online content retrieval in a large-scale CBMIR system, approximate search is usually adopted instead of the time-consuming exhaustive comparison and it can be described as follows: input a segment of a music piece, perform index-based similarity search, and finally return some relevant songs in a ranked list.

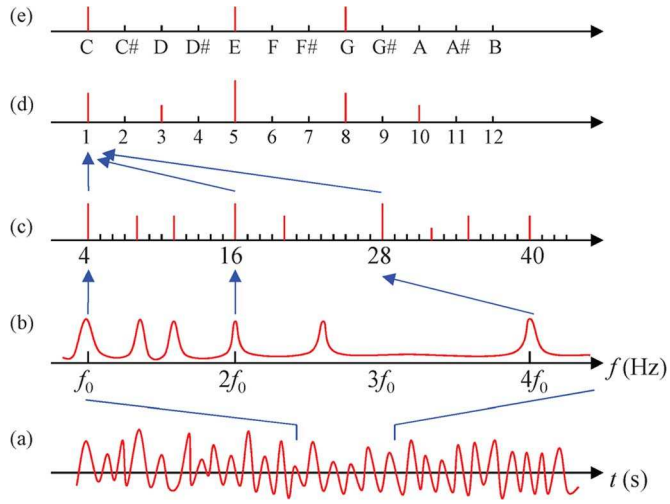

Fig. 1. Music representation: from signal to chord.

A. Representation of Music Information

An instrument-generated song can be represented at different levels. At the high level, each song has its own score. At the mid-level, at any moment, usually multiple notes are played together which correspond to a chord. At the low level, the acoustic signal of simultaneously played notes provides a specific music (spectral) perception to users.

Fig. 1 shows different levels of abstraction of music signals. A music signal in Fig. 1(a) is a long sequence of samples in the time domain. Directly comparing music signals is prohibitively expensive and finding an appropriate representation is the key to efficient music retrieval. Music signals are perceived by human beings in terms of spectral components (Fig. 1(b)). In consequence, most CBMIR algorithms rely on extracting spectral features (e.g., STFT [1], MFCC) from acoustic signals. However, it is also time-consuming to determine the similarity of two audio signals in terms of feature sequences.

The frequency resolution of the human auditory system is limited and non-linear. It is sufficient to divide the music frequency band into 88 sub-bands in the log-scale, each corresponding to a pitch note [2], [3], as shown in Fig. 1(c), where adjacent pitches are spaced by a semitone. Due to the harmonic nature of music instruments, a frequency appearing in the spectrum is often accompanied by its octave frequency. In addition, multiple pitch components are generated simultaneously in a polyphonic song. Therefore, it is relatively difficult to find the exact pitches. Harmonic frequency components are perceptually similar. Accordingly, the energy of 88 pitch sub-bands can be further grouped into 12 pitch class profiles, or the 12-dimension chroma feature [13], where energies of pitches in the same harmonic family are added together.

Different methods have been suggested to compute chroma features, e.g., chroma energy normalized statistics (CENS) in [20] and chroma DCT-reduced log pitch (CRP) in [21], where chroma is computed, smoothed and down-sampled. Beat-synchronous chroma based on instantaneous frequency (BSIF chroma) is suggested in [4], where chroma is computed per-frame using the instantaneous frequency information. Adjacent chroma features, corresponding to the same beat, are aggregated into a single feature by computing their average.

B. Chord Progression Recognition

Mid-level features of music audio tracks are compact descriptors transcribed from low-level features of acoustic signals by aid of signal processing, musical knowledge, machine learning and pattern recognition. These features capture musical attributes (e.g., chord, rhythm, and instrument) and better represent musical semantics than low-level features. They provide a brief yet accurate enough representation of audio signals and can be used as an alternative when high-level semantic features are not available. They can serve as audio signatures and help to accelerate music similarity retrieval, structure segmentation, mood classification and genre recognition.

As a mid-level feature, a chord is a concise representation of music signals. A chord [22] in music is a combination of two or more notes initiated simultaneously. Chord progression represents harmonic content and semantic structure of a music work, and influences music melody. Hence, chord recognition has attracted great interest and many efforts have been devoted to transcribing chords from music signals [13]–[18]. Major bins of a chroma feature are associated with a chord pattern, e.g., the chroma in Fig. 1(d) corresponds to major triad C in Fig. 1(e). The simplest way to chord recognition is template matching [13], computing the correlation between the chroma feature (Fig. 1(d)) and a target chord pattern (Fig. 1(e)). This, however, does not always work well since unexpected components sometimes may dominate chroma energy [14]. More advanced chord recognition involves supervised training using either a Gaussian model or a support vector machine (SVM). Instead of simple chroma [4], [20], [21], BSIF chroma, together with its quad terms, forms a compound feature with $D_O = 114$ dimensions and is used in chord recognition in [16]. In contrast to per-frame recognition, a more effective policy is to consider chord progression and exploit sequence detection by the hidden Markov model (HMM) [18].

C. Locality Sensitive Hashing

LSH [23] is an index-based data organization structure, used to quickly and approximately find items relevant to a given query. Its retrieval speed usually is much faster than that of exhaustive search algorithms. Accordingly, LSH-based methods have shown great impact on multimedia information retrieval such as music content detection [1], duplicate video mining and clustering [24], and large-scale image retrieval and searching.

Conceptually, the general idea behind LSH is very simple as follows: if items are similar to each other in the original vector space, after projecting these items into a new vector space by a family of locality sensitive hashing functions, they remain similar to each other with a high probability. In a LSH-based retrieval, at first buckets associated with the hash key determined by the query are located. Then, with a post comparison, relevant items are found and ranked according to their similarity to the query. Distribution of features in buckets tend to be uneven. This can be solved by splitting big buckets into sub-buckets [7]. Typically, many parallel LSH tables are required to achieve high retrieval accuracy, which occupies a large space. The number of hash tables can be greatly reduced by exploiting multi-probing [25], [26].

D. Related CBMIR Systems

A quick search in a large-scale music database needs a careful tradeoff between accuracy and efficiency, where retrieval efficiency can be greatly improved by summarization and LSH.

Efforts in Summarizing Audio Signals: In [27], principal component analysis (PCA) is used to compute a summary representation from typical audio features (timbre, rhythm and pitch). With annotated class information, a multi-layer neural network is trained. The activation value of each feature inside the neural network is regarded as the most discriminative information. This method is more suitable for music classification than content-based retrieval. The most-frequently-used spectral features (e.g., MFCC, chroma, and pitch) are combined into a federated feature [28] by aid of assigning a set of weights trained from a regression model. The disadvantage of these methods is that such a global feature descriptor has difficulty in retaining the local temporal information. In [29], modulated complex lapped transform (MCLT) coefficients are computed from audio samples and pass two-layer oriented PCA to generate summaries for audio segments. In [5], a multi-probe histogram (MPH) is computed from the sequence of chroma features. A histogram, over a set of predefined music concepts represented by audio words, is suggested in [30]. Local temporal information is retained in the summarization in these methods, but music knowledge is not exploited.

Organizing Music Database via LSH: The first LSH-based CBMIR system was MACSIS proposed by Yang [1]. STFT is calculated from each signal and used as the basic feature, from which hash keys are computed. Hough transform, after the indexing procedure, is performed to rank matched results in terms of sequence detection. In [31], log frequency cepstral coefficients (LFCCs) and pitch class profiles are used to represent music signals. Audio shingles, concatenating adjacent features into a high-dimensional vector, are arranged in a LSH table to support a fast, approximate nearest neighbor search. Similar results, applying random linear functions in realizing LSH, were studied in [28], [29], [32]. A sequence of consecutive pitch notes is used as a hash index in [33]. Potential errors in pitch recognition are taken into account. But this technique cannot be directly used to retrieve polyphonic songs. In [5], a two-stage LSH algorithm is exploited to improve the reliability and scalability of CBMIR systems. Besides exploiting LSH in summarizing audio signals in the first stage, MPHs are organized in the LSH table in the second stage, based on the order-statistics information.

E. Key Differences to State-of-the-Art Work

In this paper we apply musical knowledge to refine music content representation, aiming to improve both reliability and

TABLE I
TABLE OF NOTATIONS

Notations	Meaning
D_O	Dimension of chroma features
M	Number of different chords (size of chord vocabulary)
N_C	Number of probed chords per feature
N_P	Number of probed chord progressions per transition
$i \to j$	Chord progression from i to j
$\boldsymbol{z}_t$	Set of probed chord progressions at time t
c_k	Count of k^{th} chord progressions, ground truth
c'_k	Count of k^{th} chord progressions, recognized result

scalability of searching relevant songs in a large data set. In comparison with previous work, the overall benefits of our algorithm are described as follows:

- Exploiting CPs in summarizing audio signals. This heavily depends on CP recognition. But state-of-the-art algorithms [13]–[18] cannot ensure a high recognition accuracy. We solve this problem by introducing multi-probing in the SVM^{hmm} recognition, and further compute a robust mid-level feature—chord progression histogram (CPH). In this way, the summary computed by the proposed method is more accurate compared with previous works on music summarization [5], [27]–[29].
- Organizing songs in the tree-structure LSH table by using dominant CPs as hash keys. Although previous schemes [1], [7], [28], [29], [31] usually require multiple parallel hash instances, our LSH scheme only requires one hash table. Satisfactory retrieval performance is achieved by multi-probing in the search stage. We further define average degradation of ranks to refine this probing.

Multi-probing is performed in the CP recognition so as to compensate for otherwise inaccurate CPs due to the low recognition accuracy. The computed CPH is strongly associated with musical knowledge and captures most-frequent CPs, where likelihood information of each probed CP is associated with its own rank. In addition, organizing CPHs in the tree-structure LSH table according to their dominant CPs ensures that features in the same bucket are highly similar, which facilitates multi-probing in the search stage.

III. CHORD PROGRESSION-BASED RETRIEVAL ALGORITHM

In this section, we present the main retrieval algorithm. First, we describe the CBMIR framework in Section III-A, introducing the main components of the retrieval system. Directly computing the similarity between sequences of low-level features is computationally prohibitive in a large database. Therefore, we exploit CPs to compute a mid-level feature. The model used for recognizing CPs from chroma sequences and the multi-probing procedure for improving recognition accuracy are discussed in Section III-B. To avoid directly comparing two chord sequences while retaining chord progressions, we further explain how to compute a chord progression histogram (CPH) in Section III-C, focusing on how to probe CPs. Based on a similarity analysis in terms of CPs, dominant CPs are used as hash keys to design a tree-structure LSH table in Section III-D. Finally, the effect of CP recognition on LSH performance is theoretically analyzed in Section III-E. Some frequently used symbols are listed in Table I.

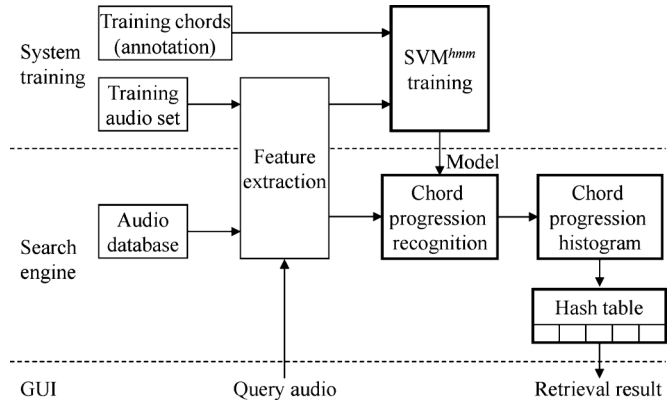

Fig. 2. Framework for a scalable CBMIR system.

A. Framework Overview

Fig. 2 shows our framework for a scalable CBMIR system. It consists of four main parts: chord model training, CP recognition, CPH computation, and hash table organization. Different classification methods are compared in [34] and SVMs showed mostly good performances. Therefore, for the training part, we use the SVM^{hmm} model [35], which considers both the spectral structure in each feature and CP embedded in adjacent features. With the trained model, CP recognition is performed for all songs in the database. Their CPHs are computed and organized in the hash table, where the set of dominant CPs of each song is used as its hash key. With a query as input, its CPs are recognized and its CPH is computed. With its hash key, relevant songs are found from the associated buckets. Finally, relevant songs are returned to the user in a ranked list as the retrieval results.

A sequence of D_O-dimensional chroma-related features is extracted from audio signal and is to be transcribed to a chord sequence. We will apply the proposed method together with several state-of-the-art features: CENS (Muller *et al.* [20], $D_O = 12$), CRP (Muller *et al.* [21], $D_O = 12$), BSIF chroma (Ellis *et al.* [4], $D_O = 12$), and CompFeat (Ellis *et al.* [16], $D_O = 114$). Distinguishing all possible chords is quite complicated. For many applications, e.g., retrieval in this paper, it is enough to use a subset of chords as the vocabulary. Similar to previous work, we mainly consider the most frequent chords: 12 major triads $(C, C\#, D, \cdots, A\#, B)$ and 12 minor triads $(c, c\#, d, \cdots, a\#, b)$. All other types of chords are regarded as one type (O). Altogether there are $M = 25$ possible chords, where $O, C, C\#, \cdots, a\#, b$ are mapped to the numbers $1, 2, \cdots, M$ respectively, so as to uniquely identify each chord.

B. N-Best Chord Progression Recognition

Each chroma feature corresponds to a chord. In addition, the composition rule of a song also places some constraints on adjacent chords, which determines CP and is reflected in adjacent features. We adopt the SVM^{hmm} model [35], SVM for per-feature chord recognition, and HMM for CP recognition.

The SVM^{hmm} model is described by (1) and explained as follows: $\boldsymbol{w}_C$ is a $M \times D_O$ matrix used to convert a $D_O \times 1$ feature to a $M \times 1$ vector of chord scores which correspond to the likelihood of chords computed from the feature (the effect of SVM). $\boldsymbol{w}_T$ is a $M \times M$ matrix describing the score of transiting from one chord to another between adjacent features (the effect of HMM). $\varphi_C(y_t)$ is a $1 \times M$ indicator vector that exactly has

only one entry set to 1 corresponding to a chord y_t. $\varphi_T(y_{t-1}, y_t)$ is a $M \times M$ indicator matrix that only has one entry set to 1 corresponding to chord progression from y_{t-1} to y_t. With a feature sequence $\{\boldsymbol{x}_t\}$ and a chord sequence $\{y_t\}, t = 1, 2, \cdots, l$, $\varphi_C(y_t) \times \boldsymbol{w}_C \times \boldsymbol{x}_t$ is the score (likelihood) that $\boldsymbol{x}_t$ is matched to chord y_t. $\varphi_T(y_{t-1}, y_t) \times \boldsymbol{w}_T$ is the score that the local chord sequence progresses from y_{t-1} to y_t. Consequently, the sum in (1) represents the total score that the feature sequence $\{\boldsymbol{x}_t\}$ is matched to the chord sequence $\boldsymbol{y} = \{y_t\}$. In the end, the chord sequence with the maximal total score is found.

$$\boldsymbol{y} = \arg\max_{\boldsymbol{y}} \left[\sum_{t=1,\cdots,l} \varphi_C(y_t) \times \boldsymbol{w}_C \times \boldsymbol{x}_t + \varphi_T(y_{t-1}, y_t) \times \boldsymbol{w}_T \right]. \tag{1}$$

Parameters $\boldsymbol{w}_C$ and $\boldsymbol{w}_T$ of the SVM^{hmm} model can be obtained by training, using the public dataset "Beatles" which has been manually annotated by Harte [36].

1) Chord Progression Recognition With Multi-Probing: Chord progression recognition by (1) only returns the chord sequence with the highest score. However, even with state-of-the-art algorithms, the chord recognition accuracy is still relatively low, which leads to a lower recognition accuracy of CPs. This is partially due to the difficulty in detecting the exact chord transition positions in music signals. When the recognized chord sequence is used for retrieval, we argue that besides the chord sequence with the highest score, other CPs should also be probed as well, in order to improve the reliability. Although new features may be suggested for improving performance of CP recognition, the multi-probing method suggested here will still work well.

The well-known Viterbi algorithm is usually used in optimal sequence detection. It simplifies sequence detection by only keeping track of one optimal path from starting point to each state at time t. It is also used in the SVM^{hmm} algorithm [35] for chord progression recognition. We modified the Viterbi algorithm shown in Algorithm 1 to realize local multi-probing, not only probing N_C chords per feature but also probing N_P CPs per transition. Actually the latter is more important in computing CPH.

Algorithm 1 Chord progression recognition

procedure CHORDPROGRECOG $(\boldsymbol{x}_t, t = 1, 2, \cdots, l)$
  $\boldsymbol{r}_1 \leftarrow \boldsymbol{w}_C \times \boldsymbol{x}_1$ ▷Initialization at $t = 1$
  $\boldsymbol{s}_1 \leftarrow \boldsymbol{r}_1$
  for $t = 2, 3, \cdots, l$ **do** ▷Forward iteration
    $r_{t,j} \leftarrow \boldsymbol{w}_{C,j} \times \boldsymbol{x}_t$
    $\boldsymbol{p}_{t,j} \leftarrow \boldsymbol{s}_{t-1} + \boldsymbol{w}_{T,j} + r_{t,j}$
  $\boldsymbol{s}_t \leftarrow [\max(\boldsymbol{p}_{t,1}), \max(\boldsymbol{p}_{t,2}), \cdots, \max(\boldsymbol{p}_{t,M})]^T$
  end for
  $\boldsymbol{y}_l \leftarrow N_C$ top chords of $\boldsymbol{s}_l$ ▷Initialization at $t = l$
  for $t = l-1, l-2, \cdots, 1$ **do** ▷Reverse iteration
    $\boldsymbol{S}_t \leftarrow \sum_{j \in \boldsymbol{y}_{t+1}} \boldsymbol{p}_{t+1,j}$
    $\boldsymbol{y}_t \leftarrow N_C$ top chords of $\boldsymbol{S}_t$
    $\boldsymbol{P}_t \leftarrow \{(i, j, \boldsymbol{p}_{t+1,j,i}) | i \in [1, \cdots, M], j \in \boldsymbol{y}_{t+1}\}$
    $\boldsymbol{z}_t \leftarrow \{(i, j, rank_{i,j}) |$ top N_P CPs of $\boldsymbol{P}_t\}$
  end for
  return $\boldsymbol{y}_t, t = 1, 2, \cdots, l$ and $\boldsymbol{z}_t, t = 1, 2, \cdots, l-1$
end procedure

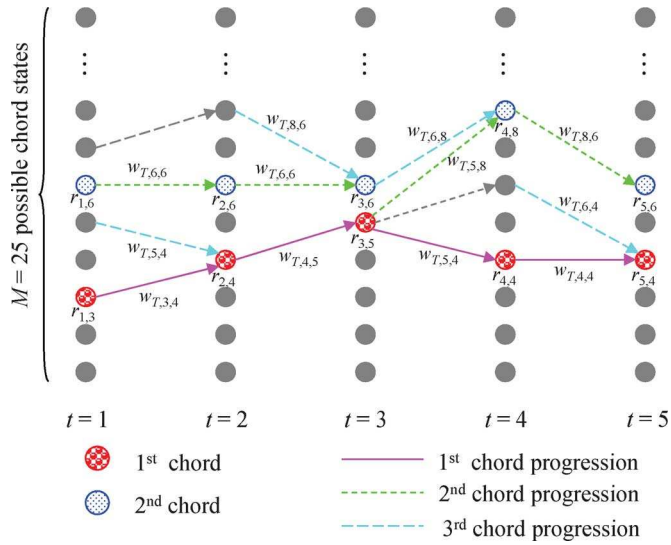

Fig. 3. Chord progression recognition with multi-probing.

This modified Viterbi algorithm takes the feature sequence $\{\boldsymbol{x}_t\}$ as input, and outputs chord set $\{\boldsymbol{y}_t\}$ and CP set $\{\boldsymbol{z}_t\}$. The procedure is divided into two parts. The first part is a forward process, where scores of all paths are computed. $\boldsymbol{r}_t = \boldsymbol{w}_C \times \boldsymbol{x}_t$ is a $M \times 1$ vector which contains scores of all chords when matched against $\boldsymbol{x}_t$. $\boldsymbol{s}_t$ is a $M \times 1$ vector, each of which corresponds to the optimal path from the beginning to a chord at t. At $t = 1$, $\boldsymbol{s}_1$ equals $\boldsymbol{r}_1$. When $t = 2, 3, \cdots, l$, scores of the paths from the beginning to chord j at t are composed of three parts: (1) $\boldsymbol{s}_{t-1}$, scores of the M optimal paths to all chords at $t-1$, (2) $\boldsymbol{w}_{T,j}$, scores of transiting from all chords at $t-1$ to chord j at t, and, (3) $r_{t,j}$, the score of chord j when matched against $\boldsymbol{x}_t$. Scores of these M paths leading to the same chord j at t are recorded in $\boldsymbol{p}_{t,j}$ and scores of the M optimal paths to M chords at t are stored in $\boldsymbol{s}_t$.

The second part is the reverse process, where potential chords and CPs are probed. At $t = l$, the N_C top chords of $\boldsymbol{s}_l$ are regarded as potential chords corresponding to the last feature. When $t = l-1, l-2, \cdots, 1$, there is a path from each chord at t to each of the N_C chords in $\boldsymbol{y}_{t+1}$ at $t+1$. Scores of these N_C paths sharing the same chord at t are added together and saved in $\boldsymbol{S}_t$, from which the top N_C chords are found as $\boldsymbol{y}_t$. The $M \times N_C$ CPs from M chords at t to N_C chords in $\boldsymbol{y}_{t+1}$ at $t+1$ form a set $\boldsymbol{P}_t$, from which the top N_P are probed. These CPs, together with their ranks, are saved in $\boldsymbol{z}_t$.

Fig. 3 shows an example with 5 features. At the t^{th} stage, the feature $\boldsymbol{x}_t$ is statistically classified to M possible chords, where the j^{th} chord has a score of $r_{t,j} = \boldsymbol{w}_{C,j} \times \boldsymbol{x}_t$. The score of progressing from chord $y_{t-1} = i$ to $y_t = j$ equals to $w_{T,i,j}$. Besides the optimal chord sequence (3, 4, 5, 4, 4) with the maximal score, there are other paths that may partly overlap with the optimal path, but with different chords and CPs somewhere. With errors occurring in chord recognition, probing sub-optimal paths becomes necessary. In Fig. 3, at each stage, $N_C = 2$ chords and $N_P = 3$ CPs are probed.

2) An Example of Chords/Chord Progressions: Fig. 4 shows some results of chord recognition with multi-probing, where $N_C = 2$ chords and $N_P = 3$ CPs are probed per stage. The beat-synchronous CompFeat feature [16] is used. The horizontal

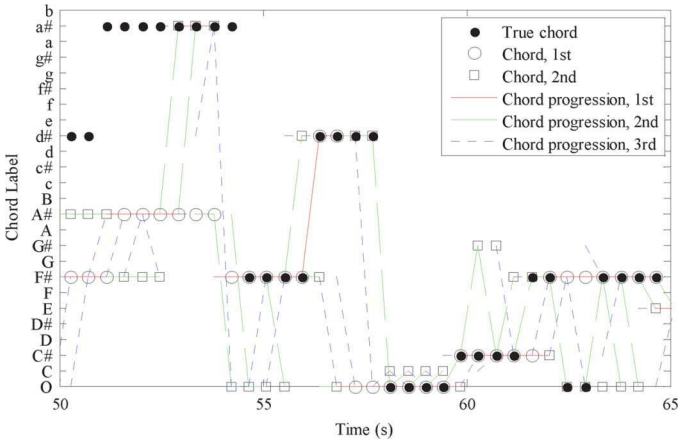

Fig. 4. Effect of chord progression recognition ("A Hard Day's Night" of the album "A Hard Day's Night" performed by "The Beatles" band).

axis is time and the vertical axis is the chord label. Solid points are true chords by manual annotation, circle and square points are recognized chords, and short lines are recognized CPs. Of the eight CPs ($d\# \rightarrow a\# \rightarrow F\# \rightarrow d\# \rightarrow O \rightarrow C\# \rightarrow F\# \rightarrow O \rightarrow F\#$), by recognition, one ($F\# \rightarrow d\#$) appears in the 1^{st} rank and five ($d\# \rightarrow O \rightarrow C\# \rightarrow F\# \rightarrow O \rightarrow F\#$) appear in the 2^{nd} rank. From 50 sec to 53 sec, $d\#$ is recognized as $F\#$, and $a\#$ is recognized as $A\#$ because they have two out of three tones in common. Correctly detecting the two chords requires more probings.

The 15 sec audio signal in Fig. 4 contains 9 chords, or 8 CPs according to annotation information. But there are many more features than chords due to the following factor: In the feature extraction stage, an audio signal is divided into short, overlapping frames. Adjacent frames corresponding to the same beat are aggregated to generate one beat-synchronous feature [16] and used for chord progression recognition. But chord boundaries cannot always be accurately detected in this stage. As a result, the same chord may span several adjacent features, e.g., $a\#$ spans 8 features. Then, CPs recognized from audio signals can be divided to two categories, *inter-chord* CPs (e.g., $d\# \rightarrow a\#$) where chords actually change, and *intra-chord* CPs (e.g., $a\# \rightarrow a\#$) where the chord is the same but divided into multiple features. Only inter-chord CPs are associated with melody information.

C. Chord Progression Histogram

The chord sequence recognized from the feature sequence is a mid-level representation of an audio signal. Directly comparing two chord sequences is faster than comparing two chroma sequences. But it still requires time-consuming dynamic programming (DP), in order to account for potential mis-alignment. To expedite the retrieval process with a more compact representation, the chord sequence can be further summarized into a chord progression histogram.

Among CPs $\{\boldsymbol{z}_t\}$ provided by Algorithm 1, each probed CP $\boldsymbol{z}_{t,k} = (i, j, rank_{i,j})$ is a triple. The progression from chord i to chord j is mapped to a CPH bin h. From the rank $rank_{i,j}$, a weight w is computed in a heuristic way so that a larger weight is used for a higher rank. More specifically, weights $N_P, N_P - 1, \cdots, 1$ are used for ranks $1, 2, \cdots, N_P$, respectively. Different weight policies are tried and by experiments we find this simple weight policy works well. The whole process is shown in (2).

$$\begin{aligned} &\text{for } t = 1, 2, \cdots, l-1, k = 1, 2, \cdots, N_P, \\ &\quad \text{get } z_{t,k} = (i, j, rank_{i,j}) \text{ from } \boldsymbol{z}_t, \\ &\quad h = (i-1) \times M + j, w = N_P - rank_{i,j} + 1, \\ &\quad CPH(h) = CPH(h) + w. \end{aligned} \tag{2}$$

In the ideal case, CPH should only include inter-chord CPs to accurately reflect the melody information. But intra-chord CPs do exist. Consider the case where two adjacent features correspond to the same chord i. By probing ($N_C = 3, N_P = 3$), N_C chords probed from the former feature are i, j, k while N_C chords probed from the latter feature are i, j, l. The N_P probed CPs are $i \rightarrow i$ (the ground truth intra-chord CP), $i \rightarrow j$ (j is probed from the latter feature) and $j \rightarrow i$ (j is probed from the former feature). Fig. 4 shows an example around 60 sec: $C\# \rightarrow G\#$ and $G\# \rightarrow C\#$ (true chord is $C\#$). In addition, there are more intra-chord CPs than inter-chord CPs, e.g., in Fig. 4 only 8 CPs are inter-chord CPs and the other 25 are intra-chord CPs. It is hard to distinguish inter-chord CPs from intra-chord CPs without knowing chord annotation. Therefore, CPs $i \rightarrow i$ are removed, but CPs $i \rightarrow j$ and $j \rightarrow i$ remain. As a result, a CPH is somewhat symmetric.

D. Tree-Structure LSH

Each song has its own CPs and two similar songs share many CPs in common. Therefore, it is possible to use CPs in the hash design. But it is necessary to first investigate to what extent two relevant songs are similar in terms of their dominant CPs. Because CPH is somewhat symmetric, we only consider CPs from i to j where $i > j$ in the hash design in order to get more unique hash keys. Altogether there are $M(M-1)/2 = 300$ possible CPs.

Let the top k CPs of two relevant songs q and d be $\boldsymbol{z}_{q,k}$ and $\boldsymbol{z}_{d,k}$, respectively. The top k CPs of song d appear in q's CP list with ranks $r_{d,q}(i), i = 1, 2, \cdots, k$. Due to errors in CP recognition, some of the top k CPs of song d might not appear in the top k CPs of song q. Instead, their ranks will be degraded. Fig. 5 shows a simple example, where the top 5 CPs (ranks $i = 1, 2, 3, 4, 5$) of song d appear in the CP list of song q with ranks $r_{d,q}(i) = 2, 3, 4, 5, 8$. To evaluate how much two songs are similar to each other in terms of their dominant CPs, we define the average degradation of ranks (ADR) between two songs d, q as follows:

$$ADR_{d,q}(k) = \frac{1}{k}\left[\sum_{i=1}^{k} |r_{d,q}(i) - i|\right]. \tag{3}$$

In the ideal case, $\boldsymbol{z}_{q,k} = \boldsymbol{z}_{d,k}$, and $ADR_{d,q}(k)$ equals 0. When the top k CPs of d appear in the CP list of q with ranks being $m+1, \cdots, m+k$, ADR equals to m. Therefore, ADR is an average degradation of ranks of CPs.

ADR can be used to assess the similarity of two songs in terms of their dominant CPs. We investigated ADR for relevant songs over a small testset. The CDF (cumulative distribution function) of ADR with $k = 5, 10, 15, 20$ are shown in Fig. 6. ADR is relatively small in most cases. In the above analysis, a CP of a song may be in any rank of its relevant songs. Usually it is only necessary to consider ranks in the top N_r CP list. By experiment, we find that setting $N_r = 3 \times k$ provides good performance. This can be used to design hash tables. If a song d

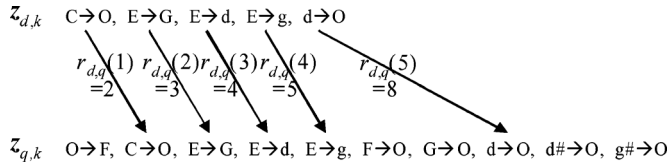

Fig. 5. Rank relationship between dominant CPs of similar songs.

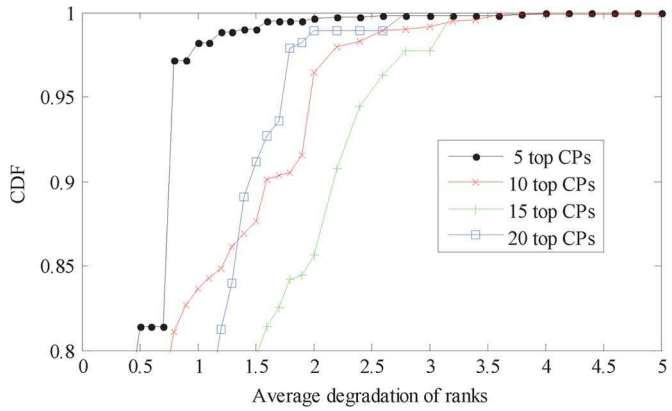

Fig. 6. Distribution of average degradation of ranks among similar songs.

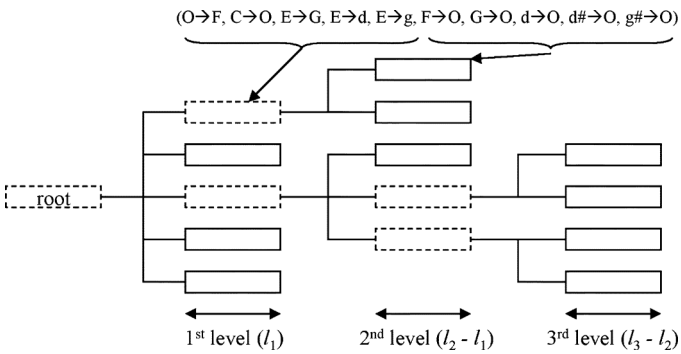

Fig. 7. Tree-structure LSH, using dominant CPs as variable-length hash keys.

has its top k CPs in the top N_r CP list of q, they are regarded as being roughly similar to each other. To further ensure a degree of similarity, $ADR_{d,q}(k)$ must be less than a pre-determined threshold, which is decided by ensuring that CDF is greater than a value, e.g., 0.98.

In the database, $\boldsymbol{z}_{d,k}$ of a song d is used as its hash key and CPH_d is stored in the associated bucket. But an investigation shows that the top k CPs are not evenly distributed. Most items are located in just a few buckets, which degrade the hashing efficiency. In this design, a large bucket is further divided into sub-buckets by using more CPs as the hash keys. For example, for a bucket with the key length being l_1, the key length of its sub-bucket will be extended to l_2. Sub-buckets belonging to the same bucket share the same prefix composed of l_1 CPs, and are distinguished by the remaining $l_2 - l_1$ CPs. In this way, the hash table is divided into n levels, and the lengths of hash keys equal to $l_1, l_2, \cdots, l_n$, respectively. Fig. 7 shows an example where $n = 3$. Dashed boxes are internal nodes representing buckets that are further divided into sub-buckets. CPHs are stored in solid boxes which correspond to leaf buckets. The longer the hash key of a leaf bucket is, the more similar songs in the same bucket will be.

The tree-structure is a little similar to the LSH forest scheme in [7]. However, the hash functions are quite different. General random hash functions, $y = ax + b \bmod p$, are used in [7], without exploiting statistical properties of features. We investigated the energy distribution of CPH and found that energy is dominated by at most 20 major bins. Using general random hash functions, most coefficients of a will not work. Therefore, many parallel hash tables are required to achieve a satisfactory recall. In our scheme, CPs are used to construct hash keys. Songs in the same bucket share the same dominant CPs. Organizing CPHs in this way ensures high similarity of songs in the same bucket. Therefore, satisfactory retrieval performance can be achieved with only one hash table, by using multi-probing [26] in the search stage.

Algorithm 2 Store CPH features in the LSH table

```
1:  procedure StoreCPH(CPH_d)        ▷CPH of song d
2:    Find top l_n CPs z_{d,l_n}
3:    Find the bucket with longest key that matches z_{d,l_n}
4:  if bucket B is found at m^th level then
5:      if B does not contain any sub-bucket then
6:        Put CPH_d in bucket B
7:        if #items in B > a threshold then
8:          DivideBucket (B, l_m → l_{m+1})
9:        end if
10:     else      ▷Target sub−bucket does not exist yet
11:       Create a new sub-bucket with the key z_{d,l_{m+1}}
12:       Put CPH_d in the sub-bucket
13:     end if
14:   else                          ▷Bucket B is not found
15:     Create a new bucket with the key z_{d,l_1}
16:     Put CPH_d in the new bucket
17:   end if
18: end procedure
19: Procedure DivideBucket(B, l_m → l_{m+1})
20:   for Each CPH_j in bucket B do
21:     Compute top l_{m+1} CPs z_{j,l_{m+1}}
22:     Put CPH_j in sub-bucket associated with z_{j,l_{m+1}}
23:   end for
24: end procedure
```

The organization of a LSH table is described by Algorithm 2. To store CPH_d in the hash table, at first its top l_n CPs $\boldsymbol{z}_{d,l_n}$ are found. Then, the bucket which has the longest key matching the one determined by CPH_d is found. Fast algorithms for searching network addresses with the longest prefix [37] can be exploited here. There are three cases: (i) This bucket exists and does not contain any sub-buckets. CPH_d is stored in the bucket. If the number of items in the bucket gets greater than a pre-determined threshold, this bucket is divided into sub-buckets using longer hash keys. (ii) A bucket is found, which is further composed of sub-buckets. But the target sub-bucket does not exist yet. A new sub-bucket is created and CPH_d is stored there. (iii) The bucket is not found. Then a new bucket is created at the first level and CPH_d is stored there.

Retrieval with the tree-structure LSH is described by Algorithm 3, for searching songs relevant to a query q, whose CPH is computed as CPH_q. Initially, the candidate set $\boldsymbol{C}_q$ is cleared to be empty. Starting from the hash level with longest hash keys, at a hash level m, from all possible l_m out of the top $3 \times l_m$ CPs of the query, the ones with $ADR_{q,q}(k)$ less than a threshold are

kept, and used as hash keys for probing. CPHs in the associated buckets are found and added to $\boldsymbol{C}_q$. Then, the similarity between CPH_q and CPHs in $\boldsymbol{C}_q$ is computed, and the set of relevant songs is returned as $\boldsymbol{R}_q$.

Algorithm 3 Retrieve items relevant to CPH_q by LSH

procedure QUERYWITHCPH(CPH_q)
▷CPH of song q
  $\boldsymbol{C}_q \leftarrow \emptyset$ ▷$\boldsymbol{C}_q$ is cleared to be empty
  $notFound \leftarrow true$
for $m = n, n-1, \cdots, 1$ **do**▷ Search each LSH level
    for all l_m CPs of top $3 \times l_m$ CPs of query **do**
      if ADR of l_m CPs $<$ a threshold **then**
        Use l_m CPs as hash key to probe buckets
        Clear $notFound$ if buckets are found
        Get CPHs from buckets and add to $\boldsymbol{C}_q$
      end if
    end for
if $notFound$ is false **then**
      Stop searching other hash levels
    end if
  end for
  Comp. similarity between CPH_q and CPHs in $\boldsymbol{C}_q$
  return the set of songs as $\boldsymbol{R}_q$ whose similarity degree
with CPH_q is greater than a threshold.
end procedure

1) Computation Cost of the Retrieval: Music composition rules constrain the possible CPs and their combinations. Assume, without loss of generality, that there are $M_t \leq M^2$ typical CPs, $CP_1, CP_2, \cdots$, with probabilities $p_1 \geq p_2 \geq \cdots$. Of the N songs in the database, $N \times \Pi_{j=1}^{l_m} p_{i_j}$ are stored in a m^{th}-level bucket with the hash key $(CP_{i_1}, CP_{i_2}, \cdots, CP_{i_{l_m}})$. The maximal number of songs in a bucket is limited to a threshold T, i.e., $N \times \Pi_{j=1}^{l_m} p_{i_j} \leq T$. To this end, a large bucket is split into sub-buckets which are assigned longer hash keys (more CPs) so that the probability product, Πp_{i_j}, is no more than T/N.

With a song as a query, the CPs used as the hash key of a m^{th}-level bucket appear in the query's CP list with ranks $r_j = j + k_j, j = 1, \cdots, l_m$, where $k_1, k_2, \cdots, k_{l_m}$ are the degraded ranks. According to (3), this bucket is probed under the following condition where ADR_{th} is the ADR threshold.

$$\sum_{j=1}^{l_m} |k_j| \leq l_m \times ADR_{th}. \tag{4}$$

The number of possible combinations of k_j, $\#k(l_m)$, determines the maximal number of probing in the m^{th} level. On the other hand, the maximal number of buckets in the m^{th} level is $C_{M_t}^{l_m}$. Therefore, the retrieval cost under LSH, normalized by the cost of exhaustive search, is limited to be no more than $\sum_{l_m} (\#k(l_m)/C_{M_t}^{l_m}) \times T/N$.

E. Performance Analysis

In this section, we give an analysis of the effect of CP recognition on the LSH design. To get a closed-form, we only consider the simple case without using different weights. Assume the false negative probability of CP recognition is p_{fn}, with which a CP is falsely recognized as other CPs. A false recognition of CP also causes a false positive event, with a probability $p_{fp} = p_{fn}/(M^2 - 1)$ under the assumption that a false CP is uniformly distributed among the other $M^2 - 1$ CPs. Consider a song with L CPs. Assume without loss of generality that the counts of $CP_1, \cdots, CP_k, \cdots CP_{M^2}$ under the ground truth equal $c_1 > \cdots > c_k > \cdots > c_{M^2}$ in the decreasing order. Their actual counts by recognition equal $c'_1, \cdots, c'_k, \cdots, c'_{M^2}$. As for CP_k,

$$c'_k = c_k - c_{fn,k} + c_{fp,k}, \tag{5}$$

where $c_{fn,k}$ is the number of CP_k falsely recognized as other CPs and $c_{fp,k}$ is the number of other CPs falsely recognized as CP_k. $c_{fn,k}$ can be modeled by a binomial distribution [38] $\mathcal{B}(c_k, p_{fn})$ and approximated by a normal distribution $\mathcal{N}(c_k \times p_{fn}, c_k \times p_{fn} \times (1 - p_{fn}))$. Similarly, $c_{fp,k}$ can be modeled by a binomial distribution $\mathcal{B}(L - c_k, p_{fp})$ and approximated by a normal distribution $\mathcal{N}((L - c_k) \times p_{fp}, (L - c_k) \times p_{fp} \times (1 - p_{fp}))$.

Therefore, c'_k approximately follows a normal distribution in (6) where $\mathcal{E}\{\}$ and $\mathcal{V}\{\}$ are operations of expectation and variance respectively.

$$\begin{aligned} c'_k &\sim \mathcal{N}(\mathcal{E}\{c'_k\}, \mathcal{V}\{c'_k\}), \\ \mathcal{E}\{c'_k\} &= c_k - c_k \times p_{fn} + (L - c_k) \times p_{fp}, \\ \mathcal{V}\{c'_k\} &= c_k \times p_{fn} \times (1 - p_{fn}) + (L - c_k) \times p_{fp} \times (1 - p_{fp}). \end{aligned} \tag{6}$$

In a similar way, we can find the distribution of c'_{k+1}. Then, $c'_k - c'_{k+1}$ also follows a normal distribution

$$\begin{aligned} c'_k - c'_{k+1} &\sim \mathcal{N}(\mathcal{E}\{c'_k - c'_{k+1}\}, \mathcal{V}\{c'_k - c'_{k+1}\}), \\ \mathcal{E}\{c'_k - c'_{k+1}\} &= (c_k - c_{k+1})(1 - p_{fn} - p_{fp}), \\ \mathcal{V}\{c'_k - c'_{k+1}\} &= \mathcal{V}\{c'_k\} + \mathcal{V}\{c'_{k+1}\}. \end{aligned} \tag{7}$$

In the LSH design, assume top k CPs are used as hash keys. Without probing, two similar songs should have their hash keys exactly the same. Then, the recognition result of their top k CPs should also be correct. In other words, all top k CPs of the ground truth results should remain top k after recognition. This can be approximated by requiring $c'_k > c'_{k+1}$ ($c'_i, i < k$ will be greater than c'_{k+1} with a higher probability), and expressed in a simple form by the often-used $\mathcal{Q}$ function $\mathcal{Q}(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{u^2}{2}\right) du$ as follows,

$$\begin{aligned} p_{noprobe} &= prob(c'_i > c'_j, \forall i \leq k, \forall j > k) \approx prob(c'_k > c'_{k+1}) \\ &= prob(c'_k - c'_{k+1} > 0), \\ &= \mathcal{Q}\left(-\frac{(c_k - c_{k+1}) \times (1 - p_{fn} - p_{fp})}{\sqrt{\mathcal{V}\{c'_k\} + \mathcal{V}\{c'_{k+1}\}}}\right). \end{aligned} \tag{8}$$

A simple probing policy is to require that top k CPs of a song appear in the top N_r CPs of its relevant audio tracks,

$$\begin{aligned} p_{probe} &= prob(c'_i > c'_j, \forall i \leq k, \forall j > N_r) \approx prob(c'_k > c'_{N_r+1}), \\ &= \mathcal{Q}\left(-\frac{(c_k - c_{N_r+1}) \times (1 - p_{fn} - p_{fp})}{\sqrt{\mathcal{V}\{c'_k\} + \mathcal{V}\{c'_{N_r+1}\}}}\right). \end{aligned} \tag{9}$$

$\mathcal{V}\{c'_k\} + \mathcal{V}\{c'_{k+1}\} \approx \mathcal{V}\{c'_k\} + \mathcal{V}\{c'_{N_r+1}\}$ and $c_k > c_{k+1} > c_{N_r+1}$. Then, $p_{probe} > p_{noprobe}$ because $\mathcal{Q}$ is a decreasing function. In other words, probing improves the retrieval recall of LSH when using dominant CPs as the hash keys.

The above simple probing policy can be further refined by setting a threshold for the ADR metric so as to only probe the most likely buckets.

IV. Evaluation

In this section we evaluate the retrieval algorithm suggested in Section III with extensive simulation evaluations. We first investigate how to select the training set for CP recognition. Then, with a small database, we examine the effects of probing on both CP recognition and retrieval, and determine the optimal parameters. Finally, we present the overall evaluation results.

A. Selecting a Training Set

State-of-the-art chord recognition algorithms, evaluated in MIREX [39], all are trained and tested on the Beatles sets [36]. About 3/4 of the 180 songs are used for training and the other 1/4 for testing. With such a policy, the trained model may be over fitted to the training set and does not generalize well to other databases.

Different from a Gaussian model which heavily depends on the size of the training set, the power of SVM comes from the support vectors themselves. The training set would work well if all typical support vectors are included. Instead of chords, we are more interested in CPs. We use the MRR1 metric to measure the performance of CP recognition. MRR1 is defined as the mean reciprocal rank of the correct CP in the probed CP list, which identifies both the recognition accuracy and the quality of CPs in times of probing. To avoid over-fitting and remove some features specific to training songs, we select a small training set from Beatles and use others as the testing set. We wish to find a small training set that contains most typical support vectors and maximizes MRR1 on the testing set with more songs so that the trained model can be better generalized to other datasets.

Algorithm 4 Find the optimal training set

procedure FINDTRAINSET$\boldsymbol{G}$
▷Set of annotated songs
  Equally divide $\boldsymbol{G}$ into N_1 groups $\boldsymbol{G}'_i, i = 1, 2, \cdots, N_1$, each with N_2 songs
for $i = 1, 2, \cdots, N_1$ **do**
    Use $\boldsymbol{G}'_i$ as the training set and train a model
    Test the model with $\boldsymbol{G} - \boldsymbol{G}'_i$, compute $MRR1'_i$
  end for
  Sort $MRR1'_i, i = 1, 2, \cdots, N_1$, in the decreasing order, accordingly $\{\boldsymbol{G}'_i\}$ becomes $\{\boldsymbol{G}_i\}$
  Use the last N_3 groups as the common testing set $\boldsymbol{T}_T$.
$\boldsymbol{T}_R \leftarrow \boldsymbol{G}_1$ and train a model
  Test it with $\boldsymbol{T}_T$ and set its $MRR1$ to $MRR1_{best}$
  for $i = 2, \cdots, N_4$ **do**
    Use $(\boldsymbol{T}_R \bigcup \boldsymbol{G}_i)$ as a temporary training set
    Train a model and test it with $\boldsymbol{T}_T$
    Compute its $MRR1$ as $MRR1_i$
    if $MRR1_i > MRR1_{best}$ **then**
      $\boldsymbol{T}_R \leftarrow \boldsymbol{T}_R \bigcup \boldsymbol{G}_i$ ▷Update the training set
      $MRR1_{best} \leftarrow MRR1_i$
    end if
    end for
    return $\boldsymbol{T}_R$ as the selected training set.
end procedure

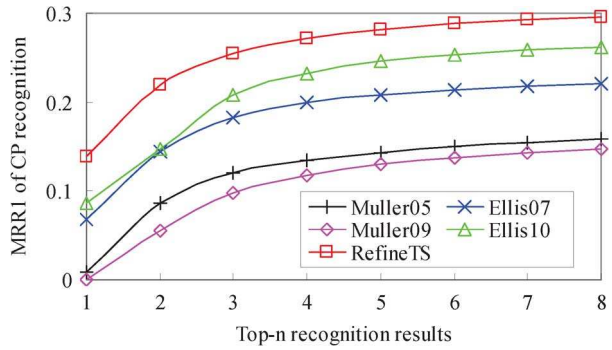

Fig. 8. Effect of multi-probing in chord-progression recognition (inter-chord CPs).

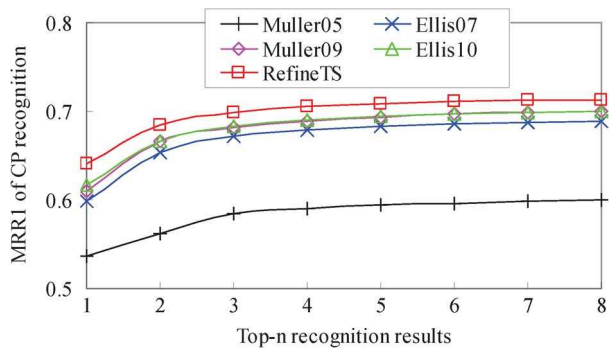

Fig. 9. Effect of multi-probing in chord-progression recognition (intra-chord CPs).

The heuristic algorithm for selecting a training set is shown in Algorithm 4, which takes as input all 180 Beatles songs ($\boldsymbol{G}$) with chord annotations, and outputs a training set $\boldsymbol{T}_R$. At first, $\boldsymbol{G}$ is divided into N_1 groups, $\boldsymbol{G}'_i, i = 1, 2, \cdots, N_1$, each with N_2 songs. N_2 should be small enough so that there will be some groups that do not contain support vectors specific to the training set. N_2 should also be large enough so that a SVM^{hmm} model can be trained. Using each group $\boldsymbol{G}'_i$ as the training set and the other songs in $\boldsymbol{G} - \boldsymbol{G}'_i$ as the testing set, $MRR1'_i$ is computed. The obtained $MRR1'_i, i = 1, 2, \cdots, N_1$, is sorted in decreasing order, and accordingly $\{\boldsymbol{G}'_i\}$ is re-arranged to $\{\boldsymbol{G}_i\}$. Then, starting with $\boldsymbol{T}_R = \boldsymbol{G}_1$ and $MRR1_{best} = MRR1_1$, a new set of songs $(\boldsymbol{T}_R \bigcup \boldsymbol{G}_i)$ is used as a temporary training set and its $MRR1$ is evaluated on the common testing set $\boldsymbol{T}_T$, and computed as $MRR1_i$. The set $(\boldsymbol{T}_R \bigcup \boldsymbol{G}_i)$ will be used as the new training set if $MRR1_i$ is greater than $MRR1_{best}$. For this process, we used $N_1 = 36, N_2 = 5, N_3 = 26, N_4 = 10$, and the final training set contains 45 songs.

B. Effect of Probing

We investigated MRR1 of chord and CPs over a common testing set, using four features referred to as Muller05 (CENS [20]), Ellis07 (BSIF chroma [4]), Muller09 (CRP [21]), and Ellis10 (CompFeat [16]). We applied the proposed multi-probing method together with all features. The recognition accuracy is usually referring to all chords. Here, we distinguish inter-chord CPs from intra-chord CPs.

MRR1 results of inter-chord CPs and intra-chord CPs are shown in Fig. 8 and Fig. 9, respectively. The two figures reveal three points: (i) The proposed multi-probing method improves

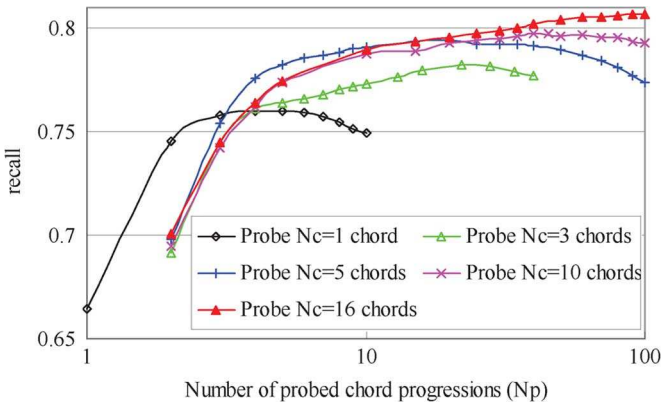

Fig. 10. Effect of multi-probing in the CP recognition on the recall performance of CBMIR.

CP recognition accuracy of all features. (ii) Intra-chord CPs are recognized with a much higher accuracy than inter-chord CPs. This is due to following factors: Spectrum within a chord is stable and the recognition accuracy of intra-chord CPs is relatively high. In contrast, spectrum near chord boundaries is not stable, which leads to a low accuracy of inter-chord CPs. Under almost all cases, the CompFeat (Ellis10) feature outperforms other features in terms of inter-chord CP accuracy. (iii) Recognition accuracy is improved by refining the training set. Together with CompFeat, we further refined the training set and another curve "RefineTS" is added to both figures, but the effect is different. For intra-chord CPs, the effect is limited because state-of-the-art algorithms already have a high performance. However, its effect on inter-chord CP accuracy is obvious. This justifies the necessity of refining the training set. The CompFeat feature with the refined model (RefineTS) is used hereafter. Further improvement of inter-chord CP recognition accuracy is left as future work.

We tried different probing policies (N_C and N_P) in computing CPHs and tested them on a small database by the kNN (k-nearest neighbor) retrieval. The result of the often-used recall metric is shown in Fig. 10. This figure reveals three points: (i) Under a fixed N_C, recall first increases with N_P and then decreases, which indicates that a suitable N_P can lead to a local maximal recall. (ii) Increasing N_C usually leads to a higher peak recall. (iii) The effect of probing is large when N_C and N_P are small. When there is no probing, $N_C = 1$ and $N_P = 1$, recall is only 0.665. Simply probing one more CP by using $N_P = 2$, the recall increases to 0.746. When probing $N_C = 16$ chords, the max recall reaches 0.806 at $N_P = 90$. This figure confirms that multi-probing in the recognition is necessary in order to get accurate music representations to improve the retrieval recall. Hereafter, $N_C = 16$ and $N_P = 90$ are used.

C. Overall Results of Retrieval Experiment

In this section, we present the overall experimental results. We use the 3 datasets shown in Table II, with a total of 74,055 audio tracks. Dataset I, Covers79, is the same as in [5] and consists of 1072 audio variants of 79 songs. More specifically, audio tracks in Covers79 are recordings of the same song sung by different people over similar music accompaniments. The proposed method can also be applied to search audio tracks with similar melodies if only they share dominant CPs in common.

TABLE II
DATASET DESCRIPTION

Datasets	Name	# Audio tracks
I	Covers79	1,072
II	Background1	10,041
III	Background2	62,942

Datasets II and III are used as background music. Since there are no large databases publicly available for simulating scalability of audio content retrieval, we collected audio tracks from MIREX, Lastfm.com, and the music channel of YouTube.

In the experiments, each track is 30 s long in mono-channel mp3 format and the sampling rate is 22.05 KHz. From these mp3 files, CompFeat [16] is calculated. Then, CPs are recognized and CPH is further computed.

We compare the proposed scheme—CPH with tree-structure LSH (CPH + TSLSH), to CPH with kNN(CPH + kNN), pitch histogram with kNN(PH + kNN), MPH with order-statistics LSH (MPH + OSLSH) [5], and CPH with LSH forest [7] (CPH + LSHForest). The LSH forest scheme is implemented with 10 parallel hash tables, each hash table using at most 20 hash functions. Its parameters (number of hash tables) are tuned so that its retrieval accuracy almost equals that of CPH + TSLSH. We also perform a comparison with a method based on the direct comparison of BSIF chroma sequences, denoted as Chroma + DP. The task is to detect and retrieve multiple items relevant to a query and rank them in an ordered list. In such tasks, recall, precision and F-measure are effective metrics. Here, relevance is assessed in terms of melody, or in other words, chord progressions.

In the following, we evaluate the precision-recall relationship, the effect of the query length and scalability with respect to the database size. Unless stated otherwise, in the evaluation, we use the following default setting: each of the 1072 tracks in Covers79 is used as the query to retrieve its relevant tracks from the datasets I + II, which have 11,113 tracks; the exception is in the third experiment where dataset III is also used for evaluating the effect of the database size. The query has the full length as their relevant tracks, except in the second experiment where the effect of the query length is evaluated. The number of ranked results equals to that of relevant items determined by the ground truth, except in the evaluation of the precision-recall relationship.

1) Precision-Recall Curves: A retrieval algorithm should make a good tradeoff between recall and precision. In this subsection we investigate this tradeoff by the classical precision-recall curves.

The number of output is changed and the pairs of recall and precision achieved by different schemes are obtained and plotted in Fig. 11. With more outputs, recall of all schemes increases because the chance that relevant songs appear in the ranked list gets larger. In contrast, precision decreases due to more non-relevant tracks. CPH + TSLSH lies between CPH + kNN and MPH + OSLSH, and is much better than PH + kNN. CPH + TSLSH is better than MPH + OSLSH mainly because music knowledge via CP recognition is exploited in CPH but not in MPH. CPH + TSLSH also outperforms CPH + LSHForest. This is because the number of hash tables in LSHForest is limited to 10. Improving recall

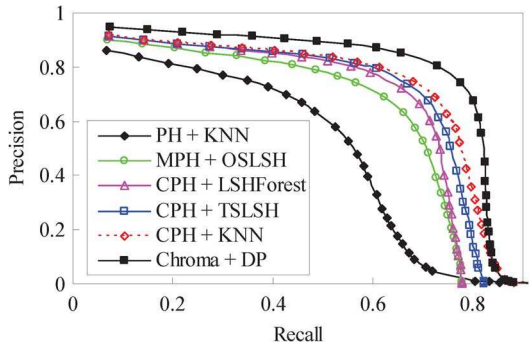

Fig. 11. Precision-recall curves of different schemes. A query audio track is used to search its relevant tracks based on melody similarity. The number of searched results is varied to get different recall-precision pairs.

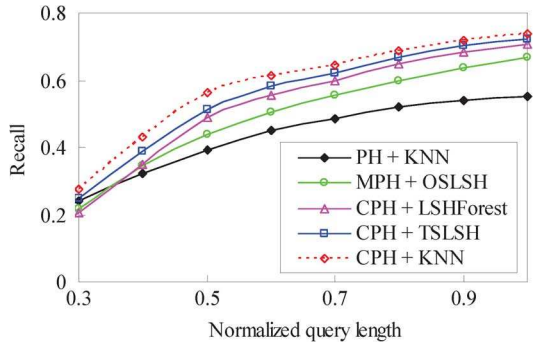

Fig. 13. Recall under different query lengths. A query audio track with a variable length is used to search its relevant tracks based on melody similarity.

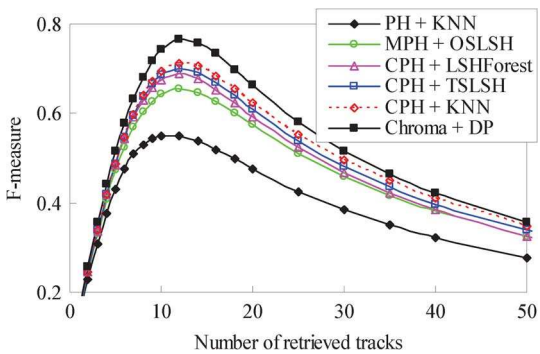

Fig. 12. F-measure of different schemes. A query audio track is used to search its relevant tracks based on melody similarity. The number of searched results is varied to get different recall-precision pairs.

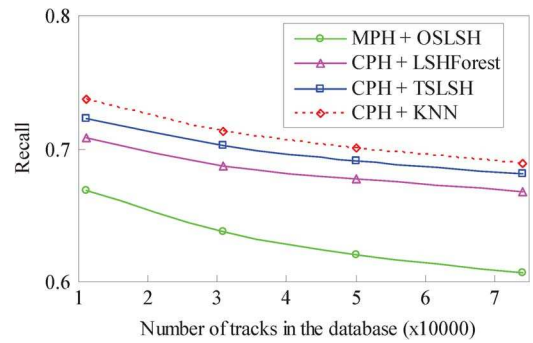

Fig. 14. Recall under different database sizes. A query audio track is used to find its relevant tracks based on melody similarity. All audio tracks have the same length. The database size is adjusted.

of LSHForest requires more hash tables. When precision equals 0.6, recall, achieved by PH + kNN, MPH + OSLSH, CPH + LSHForest, CPH + TSLSH, CPH + kNN, and Chroma + DP equal 0.5115, 0.6754, 0.7150, 0.7373, 0.7569 0.8143, respectively. Recall achieved by CPH + TSLSH is 0.077 less than that of Chroma + DP, but is 0.0223 greater than that of CPH + LSHForest and 0.062 greater than that of MPH + OSLSH.

Recall of CPH + TSLSH is satisfactory, considering that the retrieval speed is accelerated by both the summarization and indexing. CPH + TSLSH shortens the gap between MPH + OSLSH, previous efforts on global summarization and indexing technique, and Chroma + DP, which determines the upper-bound via the exhaustive sequence comparison. The performance of TSLSH is also superior to LSHForest by exploiting the non-even distribution of CPs, not to mention its much fewer hash tables. The performance difference between CPH + TSLSH and Chroma + DP is due to three factors: (i) errors in CP recognition, (ii) information loss when computing CPH from a chord sequence, and, (iii) performance loss due to approximate search by LSH. The first one is the main factor and can be alleviated by exploring more advanced features in CP recognition.

The tradeoff between precision and recall is better reflected by the F-measure metric, which combines recall and precision with the best value being 1. Fig. 12 clearly shows that the F-measure curve of CPH + TSLSH lies between those of MPH + OSLSH and Chroma + DP.

2) Effect of Query Lengths: In the last subsection, it is assumed that each query has the same length as its references in the database. However, due to various reasons, the query may be shorter. In this section, we evaluate the effect of query lengths on retrieval performance. The recall results are shown in Fig. 13, with respect to normalized query lengths.

Recall decreases with query length in all schemes. The performance is greatly degraded when the query length becomes less than 0.5. For CPH, when the query length is greater than 0.5, the recall is still satisfactory (no less than 0.5). And it is reasonable to require that the query length be no less than half of the target song in order to reliably search the relevant songs.

3) Effect of Database Sizes: LSH usually applies to large databases. By varying the database size from 11,113 to 74,055, we evaluate recall, average precision and computation cost of CPH + TSLSH.

Recall decreases in all schemes with the increase of the database size, as shown in Fig. 14. The recall difference between CPH + TSLSH and MPH + OSLSH increases from 0.0546 (database size $= 11,113$) to 0.0748 (database size $= 74,055$), indicating that CPH is more scalable with database sizes. The average precision in Fig. 15 shows a similar trend, which confirms that CPH is more accurate in representing an audio sequence than MPH. When the database size equals 74,055, recall and average precision of CPH + TSLSH equal to 0.681 and 0.878, respectively. The difference between CPH + TSLSH and CPH + LSHForest is almost irrelevant of database sizes. This is because both schemes use the same CPH feature.

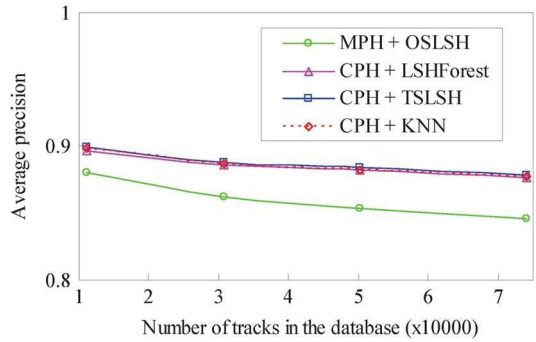

Fig. 15. Average precision under different database sizes. A query audio track is used to find its relevant tracks based on melody similarity. All audio tracks have the same length. The database size is adjusted.

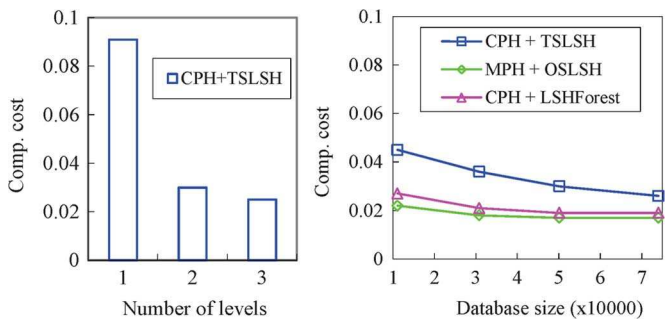

Fig. 16. Normalized computation cost in the retrieval. A query audio track is used to find its relevant tracks based on melody similarity. All audio tracks have the same length.

For the largest database size, the normalized computation, the ratio of computation cost of CPH + TSLSH to that of CPH + kNN, equals 0.091 when one-level LSH is used, it decreases to 0.03 when two-level TSLSH is used, and further decreases to 0.026 when three-level TSLSH is used, as shown in Fig. 16. The biggest gain of the tree-structure is reached when changing the LSH table from one-level to two-level. Further dividing the LSH table into three-level has little gain. This is because low-rank CPs will be used as hash keys, but their accuracy is still limited by the CP recognition algorithm. The normalized computation cost of CPH + LSHForest and MPH + OSLSH is a little less than that of CPH + TSLSH. But CPH + TSLSH achieves a better tradeoff among retrieval accuracy (recall and precision), computation cost and storage (number of hash tables).

V. Conclusions and Future Work

This paper proposes a novel method that improves accuracy and scalability of CBMIR. We have designed our retrieval algorithm by exploiting musical knowledge in training a chord model. In particular, we exploited multi-probing in CP recognition via the modified Viterbi algorithm, which outputs multiple likely CPs and increases the probability of finding the correct one. A chord progression histogram is put forward to summarize the probed CPs in a concise form, which is both efficient and also retains local chord progressions. Average degradation of ranks is suggested as a metric to assess similarity of two songs in terms of their CPs. Hash keys are also based on CPs. By setting an ADR threshold, it is possible to only probe buckets in which songs are highly similar to the query, and the number of probings is controlled. In addition, the tree structure LSH enables a more efficient organization of the database. After conducting extensive experiments looking at recall/precision curves, effect of query lengths, and scalability of database sizes, we confirmed that CPH + TSLSH is superior to previous work in terms of the tradeoff between accuracy and efficiency over a large-scale real web audio dataset.

Currently, the retrieval performance of CPH + TSLSH is still limited by the CP recognition accuracy. This could be solved by improving the accuracy of inter-chord CPs and reducing the negative effects of intra-chord CPs. This is left as future work.

References

[1] C. Yang, "Efficient acoustic index for music retrieval with various degrees of similarity," in *Proc. ACM MM*, 2002, pp. 584–591.
[2] W. H. Tsai, H. M. Yu, and H. M. Wang, "A query-by-example technique for retrieving cover versions of popular songs with similar melodies," in *Proc. ISMIR*, 2005, pp. 183–190.
[3] R. Miotto and N. Orio, "A methodology for the segmentation and identification of music works," in *Proc. ISMIR*, 2007, pp. 239–244.
[4] D. Ellis and G. Poliner, "Identifying cover songs with chroma features and dynamic programming beat tracking," in *Proc. ICASSP*, 2007, pp. 1429–1432.
[5] Y. Yu, M. Crucianu, V. Oria, and E. Damiani, "Combing multi-probing histogram and order-statistics based LSH for scalable audio content retrieval," in *Proc. ACM MM*, 2010, pp. 381–390.
[6] T. E. Ahonen, "Combing chroma features for cover version identification," in *Proc. ISMIR*, 2010, pp. 165–170.
[7] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: Self tuning indexes for similarity search," in *Proc. WWW*, 2005, pp. 651–660.
[8] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 128–131, 2008.
[9] B. Cui, J. Shen, G. Cong, H. Shen, and C. Yu, "Exploring composite acoustic features for efficient music similarity query," in *Proc. ACM MM*, 2006, pp. 634–642.
[10] I. Karydis, A. Nanopoulos, A. N. Papadopoulos, and Y. Manolopoulos, "Audio indexing for efficient music information retrieval," in *Proc. MMM*, 2005, pp. 22–29.
[11] J. Shen, D. Tao, and X. Li, "QUC-tree: Integrating query context information for efficient music retrieval," *IEEE Trans. Multimedia*, vol. 11, no. 2, pp. 313–323, 2009.
[12] N. Bertin and A. Cheveigne, "Scalable metadata and quick retrieval of audio signals," in *Proc. ISMIR*, 2005, pp. 238–244.
[13] T. Fujishima, "Realtime chord recognition of musical sound: A system using common Lisp music," in *Proc. ICMC*, 1999, pp. 464–467.
[14] K. Lee, "Automatic chord recognition from audio using enhanced pitch class profile," in *Proc. ICMC*, 2006.
[15] H.-T. Cheng, Y.-H. Yang, Y.-C. Lin, I.-B. Liao, and H. H. Chen, "Automatic chord recognition for music classification and retrieval," in *Proc. ICME*, 2008, pp. 1505–1508.
[16] D. Ellis and A. Weller, "The 2010 LABROSA chord recognition system," in *Proc. MIREX*, 2010.
[17] T. Cho, R. J. Weiss, and J. P. Bello, "Exploring common variations in state of the art chord recognition systems," in *Proc. Sound and Music Computing Conf.*, 2010.
[18] M. McVicar, Y. Ni, T. D. Bie, and R. S. Rodriguez, "Leveraging noisy online databases for use in chord recognition," in *Proc. ISMIR*, 2011, pp. 639–644.
[19] Y. Yu, R. Zimmermann, Y. Wang, and V. Oria, "Recognition and summarization of chord progressions and their application to music information retrieval," in *Proc. IEEE ISM*, 2012, pp. 9–16.
[20] M. Muller, F. Kurth, and M. Clausen, "Audio matching via chroma-based statistical features," in *Proc. ISMIR*, 2005, pp. 288–295.
[21] M. Muller, S. Ewert, and S. Kreuzer, "Making chroma features more robust to timbre changes," in *Proc. ICASSP*, 2009, pp. 1877–1880.
[22] Chord. [Online]. Available: http://en.wikipedia.org/wiki/Chord_(music).
[23] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. ACM STOC*, 1998.
[24] S. Poullot, M. Crucianu, and O. Buisson, "Scalable mining of large video databases using copy detection," in *Proc. ACM MM*, 2008, pp. 61–70.
[25] A. Joly and O. Buisson, "A posteriori multi-probe locality sensitive hashing," in *Proc. ACM MM*, 2008, pp. 209–218.

[26] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proc. VLDB*, 2007, pp. 950–961.
[27] J. Shen, J. Shepherd, and A. Ngu, "Towards effective content-based music retrieval with multiple acoustic feature combination," *IEEE Trans. Multimedia*, vol. 8, no. 6, pp. 1179–1189, 2006.
[28] Y. Yu, J. S. Downie, L. Chen, K. Joe, and V. Oria, "Searching musical datasets by a batch of multi-variant tracks," in *Proc. ACM MIR*, 2008, pp. 121–127.
[29] R. Cai, C. Zhang, L. Zhang, and W.-Y. Ma, "Scalable music recommendation by search," in *Proc. ACM MM*, 2007, pp. 1065–1074.
[30] J. Shen, H. Pang, M. Wang, and S. Yan, "Modeling concept dynamics for large scale music search," in *Proc. ACM SIGIR*, 2012, pp. 455–464.
[31] M. Casey, C. Rhodes, and M. Slaney, "Analysis of minimum distances in high-dimensional spaces," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 16, no. 5, pp. 1015–1028, 2008.
[32] R. Miotto, "Content-based music access: an approach and its applications," in *Proc. FDIA*, 2009, pp. 69–75.
[33] Z. Guo, Q. Wang, G. Liu, and J. Guo, "A query by humming system based on locality sensitive hashing indexes," *Signal Process.,* 2012. [Online]. Available: http://dx.doi.org/10.1016/j.sigpro.2012.09.006.
[34] D. Meyer, F. Leisch, and K. Hornik, "The support vector machine under test," *Neurocomputing*, vol. 55, no. 1–2, pp. 169–186, 2003.
[35] T. Joachims, T. Finley, and C.-N. Yu, "Cutting-plane training of structural SVMs," *Mach. Learn. J.*, vol. 77, no. 1, pp. 27–59, 2009.
[36] C. Harte and M. Sandler, "Automatic chord identification using a quantized chromagrams," in *Proc. Convention Audio Engineering Society*, 2005.
[37] M. A. R. Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Netw. Mag.*, vol. 15, no. 2, pp. 8–23, 2001.
[38] D. P. Bertsekas and J. N. Tsitsiklis*, Introduction to Probability*. Belmont, MA: Athena Scientific, 2002.
[39] MIREX. [Online]. Available: http://www.music-ir.org/mirex/wiki/2011:Audio_Chord_Estimation.

Yi Yu received the Ph.D. degree in computer science in 2009 from Nara Womens University. She worked at different institutions including New Jersey Institute of Technology, University of Milan and Nara Womens University. She currently works at School of Computing, National University of Singapore. Her research interests include social interactions over geo-aware multimedia streams, multimedia/music signal processing, audio classification and tagging, locality sensitive hashing-based music information retrieval, and pest sound classification. She received a best paper award from IEEE ISM 2012.

Roger Zimmermann (S'93–M'99–SM'07) received the M.S. and Ph.D. degrees from the University of Southern California (USC) in 1994 and 1998. He is currently an associate professor in the Department of Computer Science at the National University of Singapore (NUS). He is also a deputy director with the Interactive and Digital Media Institute (IDMI) at NUS and a co-director of the Centre of Social Media Innovations for Communities (COSMIC). His research interests are in the areas of streaming media architectures, distributed and peer-to-peer systems, mobile and geo-referenced video management, collaborative environments, spatio-temporal information management, and mobile location-based services. He has coauthored a book, six patents, and more than 150 conference publications, journal articles, and book chapters. He is a member of ACM.

Ye Wang (M'99) is an Associate Professor in the Computer Science Department at the National University of Singapore (NUS) and NUS Graduate School for Integrative Sciences and Engineering (NGS). He established and directed the sound and music computing (SMC) Lab. Before joining NUS he was a member of the technical staff at Nokia Research Center in Tampere, Finland for 9 years. His research interests include sound analysis and music information retrieval (MIR), mobile computing, and cloud computing, and their applications in music edutainment and e-Health, as well as determining their effectiveness via subjective and objective evaluations. His most recent projects involve the design and evaluation of systems to support 1) therapeutic gait training using Rhythmic Auditory Stimulation (RAS), and 2) Melodic Intonation Therapy (MIT). In the academic year 2011–2012 he took his sabbatical leave at the School of Computer Science of Fudan University and at Harvard Medical School.

Vincent Oria is an associate professor of computer science at the New Jersey Institute of Technology. His research interests include multimedia databases, spatio-temporal databases and recommender systems. He has held visiting professor positions at various institutions including National Institute of Informatics (Tokyo, Japan), ENST (Paris, France), Universit de Paris-IX Dauphine (Paris, France), INRIA (Roquencourt, France), CNAM (Paris, France), Chinese University of Hong Kong (Hong Kong China) and the Universit de Bourgogne (Dijon, France).