Scalable Music Recommendation by Search

Rui Cai, Chao Zhang, Lei Zhang, and Wei-Ying Ma
Microsoft Research, Asia
49 Zhichun Road, Beijing 100080, P.R. China
{ruicai, v-chaozh, leizhang, wyma}@microsoft.com

ABSTRACT

The growth of music resources on personal devices and Internet radio has increased the need for music recommendations. In this paper, aiming at providing an efficient and general solution, we present a search-based solution for scalable music recommendations. In this solution a music piece is first transformed to a music signature sequence in which each signature characterizes the timbre of a local music clip. Based on such signatures, a scale-sensitive method is then proposed to index the music pieces for similarity search, using the locality sensitive hashing (LSH). The scale-sensitive method can numerically find the appropriate parameters for indexing various scales of music collections, and thus can guarantee a proper number of nearest neighbors are found in search. In the recommendation stage, representative signatures from snippets of a seed piece are extracted as query terms, to retrieve pieces with similar melodies for suggestions. We also design a relevance-ranking function to sort the search results, based on the criteria that include *matching ratio*, *temporal order*, *term weight*, and *matching confidence*. Finally, with the search results, we propose a strategy to generate a dynamic playlist which can automatically expand with time. Evaluations of several music collections at various scales showed that our approach achieves encouraging results in terms of recommendation satisfaction and system scalability.

Categories and Subject Descriptors

H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*methodologies and techniques, systems*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*retrieval models, search process*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*indexing methods*

General Terms

Algorithms, Performance, Experimentation

MM'07, September 23–28, 2007, Ausburg, Bavaria, Germany.

Keywords

Scalable music recommendation, automated playlist generation, content-based music search, music signature, music snippet, locality sensitive hashing (LSH)

1. INTRODUCTION

Music is more pervasive than ever as an important entertainment medium. With the increase in online music stores and services, and the popularization of portable music devices, we can more conveniently access music than ever before. For example, we can purchase compact discs from electronic marketplaces such as iTunes and Amazon.com, and listen to Internet radio services such as Pandora [3] and Last.fm [1], or enjoy music anywhere with portable mp3 players like iPod and Zune. All these channels, including portable devices, provide massive music resources. For example, a typical mp3 player with 30GB hard disk can hold more than $5,000$ songs. With such a vast scale of collections, a "long tail" distribution can be observed in user listening history. That is, in their collections, except for a few pieces that are frequently played, most are rarely visited, due to the inconvenient operating functions of portable devices. Even on desktop computers, it is usually a tedious task to select a group of favorite pieces from a larger music collection. Therefore, music recommendations are highly desired because users a perceived need for suggestions to find and organize pieces close to their tastes.

Much research effort has been devoted to music recommendations in recent years [4,10,13,16,18,19,23–25,27]. Two major recommendation technologies have been reported in the literature: collaborative filtering (CF)-based and content-based recommendation. CF-based methods [10] recommend music by investigating user group ratings history, and are widely adopted in online stores and music societies [1]. To achieve reasonable suggestions, CF-based methods should be based on large-scale rating data and an adequate number of users. However, it is difficult to extend CF-based methods to applications like recommendations on local personal music collections. Moreover, CF-based methods still suffer from problems like lack of data and poor varieties of recommendation results [18, 27].

Content-based methods can meet the requirements of more application scenarios, since they focus on the properties of the music itself. Content-based methods can be further divided into metadata-based [4, 23–25] and acoustic-based methods [13, 16, 18, 19, 27]. Metadata, which includes properties such as artists, genre, and track title, are global catalog attributes supplied by music publishers. Based on such attributes, some criteria or constraints can be set up to fil-

ter favorite pieces. However, building optimal suggestion sequences based on multiple constraints is an NP-hard problem [4]. Although acceleration algorithms like simulated annealing [23] have been proposed, it is still difficult to extend such methods to a scale with thousands of pieces and hundreds of constraints. Other methods based on metadata, utilize statistical learning to construct recommendation models from existing playlists [24, 25]. Due to the limitation of training data, such learning-based approaches are also difficult to scale up. Furthermore, metadata is too coarse to describe and distinguish individual pieces' characteristics. And in practice, it is difficult to obtain complete and accurate metadata in most situations.

In comparison, acoustic-based methods have far few restrictions and are more suitable for situations in which consumers or service providers own the music themselves [3]. In general, acoustic-based methods first extract some physical features from audio signals, and then construct distance measurements or statistical models to estimate the similarity of two music objects in the acoustic space. In recommendation, music pieces with similar acoustic characteristics are grouped as suggestion candidates. For example, Knees *et al.* [16] extracted Mel frequency cepstral coefficients (MFCCs) on short-time audio segments, and each music track is then modeled using a Gaussian mixture model (GMM), based on which pair-wise distances between pieces are finally computed. In [13], music tracks are grouped using the LBG quantization based on MPEG-7 audio features, and the group closest to the seed piece is returned as suggestions. Similar to [13], Logan [19] construct music clusters using MFCCs and K-means. And, in [8], some mid-level acoustic descriptions like tempo, meter, and rhythm patterns are computed, and a global measure of similarity is then defined by assigning specific weights to these diverse feature dimensions.

By investigating various recommendation scenarios, we found that scales of music collection are quite different. For example, a music fan needs help to automatically create an ideal playlist from hundreds of pieces on an iPod; while an online music radio provider should do the same job but from several million pieces. However, almost all the aforementioned methods on music recommendation face the problem of scalability, either when scaling down or scaling up. CF-based methods must rely on large-scale user data, and performance will decrease significantly when the data scale drops. Content-based approaches mainly use linear scan to find candidates for suggestions, and processing time will increase linearly with the data scale. To accelerate the processing time on large-scale music collections, most content-based approaches utilize track-level descriptions of pieces, i.e., a whole music piece is characterized with one feature vector [4, 8, 23, 24] or one model [16]. Some approaches further group music pieces into clusters, and the similarity search is carried out on the cluster-level [13, 19]. To our knowledge, the best performance reported in one state-of-the-art work is tenths of a second for one match over a million pieces [8]. Although the processing speed is improved, such high-level descriptions may not be able to provide enough information to characterize and distinguish various pieces. On the one hand, music is a time sequence and the temporal characteristics should be taken into account when estimating the content similarity. On the other, some high-level descriptions are too coarse and are incapable of filtering an ideal suggestion from many similar candidates. Furthermore, another disadvantage of current approaches is that they are bound to given music collections, and are basically grounded on pre-computed pair-wise similarities. Therefore, update costs are considerable. While in real situations, the members of a music collection usually change frequently, especially in personal collections.

A perfect solution should combine all the advantages of CF-, meta-, and acoustic-based methods. However, in this paper we mainly focus on the problem of acoustic-based music recommendation, and leave the multi-modality recommendation problem as future work. Specifically, we try to provide a scalable scheme to meet recommendation requirements on various scales of music collections. The main idea here is to convert the recommendation problem to a scalable search problem, or, in brief, recommendation-by-search. Actually, web search can be considered a kind of recommendation. That is, users submit requests (queries) and the recommender (search engines) returns suggestions (web pages). Analogously, a musical piece can be regarded as a web page, and can be indexed based on its local melody segments (just like a web page is indexed based on keywords) for efficient retrieval.

Compared with current methods, recommendation-by-search has the following advantages. First, search technologies have been proven efficient and can be scaled from a local desktop, intranet, to the entire Web. Second, as users select and organize queries (recall the scenario of query-by-humming (QBH) [17], users decide which part of a piece they will hum as a query), user interaction can be seamlessly integrated into search-based recommendation. We noticed that a similar idea has been revealed in [2]. Moreover, updating is more convenient and cheaper by means of the search-based approach. We only need to incrementally update the index, without the need to go through the whole music collection to re-estimate pair-wise similarities.

However, for the purpose of scalable music recommendation, there are still several issues that should be addressed:

- First and the most important, is how to design a proper index structure based on data scale. Under different data scales, the criterion of "similarity" between music segments should be adaptively changed to guarantee a proper number of candidates retrieved for suggestion.

- Second, how to prepare seeds for query is another key problem in recommendation-by-search. As mentioned before, it may be no a good idea to take an entire musical piece as a seed, since in most cases only parts of sections in a piece impress users.

- Third, with a list of retrieval results, a well-designed strategy is required to rank these results based on their similarities to the seed, and finds the most appropriate one for recommendation. This is similar to the dynamic ranking of resulting pages in web search.

To construct a search-based system for scalable music recommendation, we propose a solution to address the above problems. In the proposed solution, a musical piece is first represented with a music signature sequence in which the signature characterizes one local music segment. Then, the local sensitive hashing (LSH) method [11, 14] is applied to index signatures to consider their L_2 distances. To address the first problem, we propose an algorithm to adaptively estimate appropriate parameters for LSH indexing on a given scale of the music collection. In recommendation, we extract representative signatures as query terms from a seed piece by

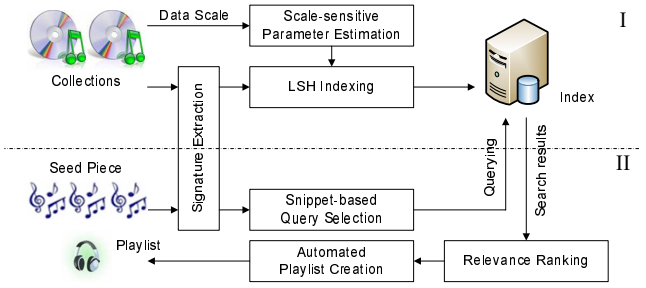

Figure 1: The flowchart of the proposed solution for scalable music recommendation, which consists of two parts: (I) scale-sensitive music indexing; and (II) recommendation-by-search.

using a music snippet analysis [5,20] and design a relevance-ranking function to integrate criteria such as *matching ratio*, *temporal order*, *term weight*, and *matching confidence*. In addition, we also propose a strategy for dynamic playlist generation based on the search results, in which the requirements of "stick to the seed" and "drift for surprise" [22] are well balanced. Finally, we evaluate our approach on various collections from around 1,000 pieces to more than 100,000 pieces, and the experimental results showed that our approach achieves promising performance on both recommendation satisfaction and system scalability, with relatively low CPU and memory costs.

The rest of this paper is organized as follows. In Section 2, an overview of the proposed approach is presented. In Section 3, we introduce our method and implementation for scale-sensitive music indexing. Section 4 describes the detailed process of the recommendation-by-search, and how to automatically construct a playlist using the proposed method. Evaluations and discussions are given in Section 5. In the last section, we draw conclusions and point out future research directions.

2. OVERVIEW OF OUR APPROACH

The flowchart of our solution is illustrated in Figure 1, which mainly consists of two parts: scale-sensitive music indexing and recommendation-by-search.

In the indexing stage, we first extract a sequence of signatures for each piece in the music collection, just like the process of term extraction in text document indexing. Here, a signature is a compact representation of a short-time music segment based on low-level spectrum features. With a signature sequence, the local spectral characteristics and their temporal variation over a music piece are preserved so as to provide more information than previous track-level descriptions. All these signatures are then organized by inverted indexes based on hash codes generated by LSH. We utilize LSH because it theoretically guarantees signatures that are close to one another will fall into the same hash-bucket with high probability. A key problem here is how to define the criterion of "*closeness*" in LSH, which directly affects the system performance. In our solution, we automatically estimate such a "*closeness*" boundary based on the scale of a music collection, to ensure a proper number of results can be retrieved for recommendation. Intuitively, the boundary of such "*closeness*" in indexing should be somewhat relaxed for a small music collection and tightened for a massive collection.

In the recommendation stage, the seed piece is also converted to a signature sequence, based on which snippets of the piece are extracted. Snippets (or thumbnails) are those representative segments in a music piece, and are usually the main chorus or highlights parts. Thus, we select signatures from the snippets instead of from the whole piece, to construct queries for retrieval. The returned search results are then sorted through a relevance-ranking function. In the ranking function, besides some sophisticated criteria widely used in text search, we also introduce several new criteria to meet the specialties of music search. Finally, we construct a dynamic playlist using the ranked search results.

In the implementation, we build an efficient disk-based indexing storage, and only a small cache is dynamically kept in memory to speed up the search process. In this way, our system can operate on most off-the-shelf PCs.

3. SCALE-SENSITIVE MUSIC INDEXING

In this section, we describe in detail the proposed method and the implementation of the scale-sensitive music indexing. This is the offline part of the proposed scheme.

3.1 Music Signature Generation

Some work related to music signature extraction has been discussed in the literature on audio fingerprinting [6,7]. However, the fingerprints defined in various articles are quite different from each other. For example, some are based on the distortion between two adjacent 10-ms audio frames and some are based on the statistics of a whole audio stream [7]. In this work, we adopt a method similar to the two-layer oriented principal component analysis (OPCA) presented in [6], as it is based on a length suitable for our requirement and is robust enough to overcome noise and distortions caused by music encoding.

In the implementation, all music files are converted to 8 kHz, 16-bit, and mono-channel format, and are divided into frames of 25.6 ms with 50% overlapping. For each frame, 1024 modulated complex lapped transform (MCLT) coefficients [21] are first computed and are then transformed to a 64-dimensional vector through the first-level OPCA. Further, to characterize the temporal variation, such 64-dimensional vectors from 32 adjacent frames (around 4.2 seconds) are concatenated and again transformed to a new 32-dimensional vector through the second-level OPCA. Here, the MCLT coefficients are used to describe the timbre characteristics on spectrum for each frame; and the time window is experimentally selected as 4.2 seconds to characterize the trend of temporal evolution. In this way, both spectral and temporal information of the corresponding audio segment is embedded in the last 32-dimensional vector, which is taken as a signature in our work. Thus, a piece is finally converted to a sequence of signatures by repeating the above operation through the whole audio stream.

3.2 Music Indexing

The objective of music indexing is to build an efficient data structure to accelerate similarity search. It is worth noticing that the music indexing in this work is quite different to those introduced in audio fingerprinting related works. In fingerprinting systems, the key difference is that only identical fingerprints are allowed to be indexed together, and two fingerprints with only small differences may have quite different index references [8,25]. In our approach, we use similarity search and try to group those close signatures in the

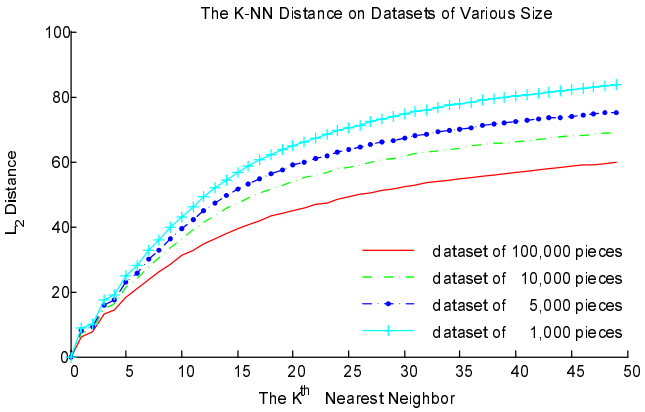

Figure 2: The average K-NN L_2 distances on four music collections with various scales.

indexing. We also investigate how to control the tolerance of such "*closeness*" to ensure a proper number of signatures can be indexed together in the same hash bucket.

3.2.1 Locality Sensitive Hashing

Locality sensitive hashing (LSH) was first proposed in [14] and was extended in [11,15], as an efficient approach to solve the problem of high-dimensional nearest neighbor search. LSH is based on a family of hash functions $\mathcal{H} = \{h : S \rightarrow U\}$, which is called locality sensitive for the distance function $D(\cdot,\cdot)$, if and only if for any $p, q \in S$, it satisfies:

$$Pr_{\mathcal{H}}(h(p) = h(q)) = f_D(D(p,q)) \quad (1)$$

where $f_D(D(p,q))$ is monotonically decreasing with $D(p,q)$. Given a (R,λ,γ)-high dimensional nearest neighbor search problem [1], LSH uniformly and independently selects $L \times K$ hash functions from $\mathcal{H}$, and hashes each point into L separate buckets. Thus two closer points will have higher collision probabilities in the L buckets. More details can be found in [11]. It has been theoretically proven that given a certain (R,λ,γ), the optimal L and K can be automated estimated [15]. In the nearest neighbor search problem, the probabilities λ and γ can be experientially selected, and the last problem is how to select a proper R.

The value of R directly affects the expectation of how many neighbors can be retrieved with probability λ using LSH. Figure 2 shows such an example. In Figure 2, we randomly sampled 1000 signatures as query terms from four music collections with different scales [2], respectively, and then compute the average L_2 distance of a term to its K^{th} neighbor for each collection. From Figure 2, it is clear that to return a given number of neighbors, different boundaries should be set for different data scale. Intuitively, such a boundary should be relaxed for small set while tightened when data scale increases, to ensure an expected number of neighbors can be returned. Specifically, it a requirement of recommendation-by-search is to promise a proper number of pieces will be returned for suggestion on whatever scale of music collections.

[1]That is, for any given query point q, each point p satisfying $D(p,q) \leq R$ should be retrieved with probability at least λ, and each point satisfying $D(p,q) > R$ should be retrieved with probability at most γ [15].

[2]The detailed information of the four collections please refers to Section 5.1.

Table 1: The mean μ and standard deviation σ of the pair-wise distances of signatures on various scales of music collections.

Scale ≈	1,000	5,000	10,000	100,000
μ	177.1	176.3	175.8	175.6
σ	39.3	39.3	39.2	39.2

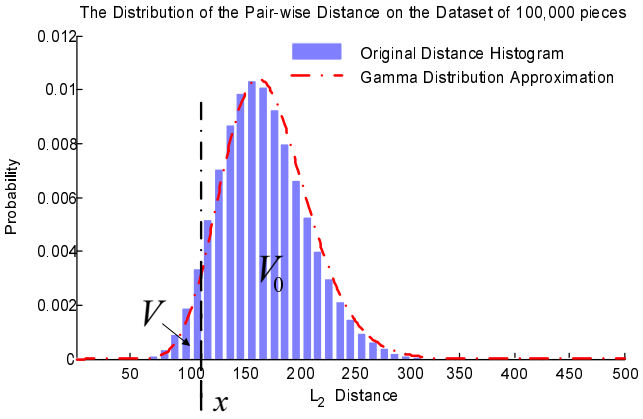

Figure 3: The distribution of the pair-wise distance of music signatures on a collection with more than $100,000$ pieces. It can be approximated using a Gamma distribution.

3.2.2 Scale Sensitive Parameter Estimation

In this work, we propose a numerical method to automatically estimate the value of R for a given scale of music collection. An assumption here is, whatever the data scale is, the distribution of the pair-wise L_2 distances among signatures should be relatively stable. A similar conclusion has been drawn in [9]. To verify such an assumption, we checked the pair-wise distances on the four collections in our experiments, and list the corresponding mean μ and standard deviation σ in Table 1.

From Table 1, it is clear that the means and standard deviations of the pair-wise distances are close on various scales of the collections. We also draw the histogram of the distance distribution on the collection that contains more than 100,000 pieces in Figure 3. Such a distribution is similar to a Gaussian distribution. However, it is asymmetric since the L_2 distance is always larger or equal to zero, and it can be better approximated by a Gamma distribution, as shown in Figure 3. The probability density function (*pdf*) of a Gamma distribution is:

$$g(t;\alpha,\theta) = t^{\alpha-1}\frac{e^{-t/\theta}}{\Gamma(\alpha)\theta^{\alpha}} \quad (2)$$

where the two parameters α and θ can be estimated as:

$$\alpha = \mu^2/\sigma^2; \quad \theta = \sigma^2/\mu \quad (3)$$

Based on the above assumption, we can consider that for various music collections, the pair-wise L_2 distances of our signatures follow a same Gamma distribution $g(t;\alpha,\theta)$. Thus, given the data scale V_0 and the expected result number V, the optimal value of R can be obtained by solving the following equation (R is replaced by x for clarity):

$$f(x) = \int_0^x g(t;\alpha,\theta)\,dt - \rho = \int_0^x t^{\alpha-1}\frac{e^{-t/\theta}}{\Gamma(\alpha)\theta^{\alpha}}\,dt - \rho \quad (4)$$

and $\rho = V/V_0$ is the expected ratio of the returned results. In our experiments, V is set to 20 for all the datasets. Let $s = t/\theta$, equation (4) is further transformed to:

$$f(x) = \frac{1}{\Gamma(\alpha)} \int_0^{x/\theta} s^{\alpha-1} e^{-s}\, ds - \rho = \frac{1}{\Gamma(\alpha)} \gamma(\alpha, \frac{x}{\theta}) - \rho \quad (5)$$

where $\gamma(\alpha, x)$ is a lower incomplete Gamma function, and can be solved numerically. Thus, x can be iteratively achieved using the Newton-Raphson method with a random initial value x_0, as:

$$x_{n+1} = x_n - f(x_n)/f'(x_n) \quad (6)$$

where the derivative $f'(x) = g(t; \alpha, \theta)$.

In this way, we can estimate a proper R and construct a LSH-based index, according to the scale of a given music collection. In the search stage, a query signature is hashed by the same set of LSH hash functions, and its neighbors are independently retrieved from the corresponding L buckets.

4. RECOMMENDATION-BY-SEARCH

Music in a similar style usually adopts some typical rhythm patterns and instruments. For example, fast drum beat patterns are widely used in most heavy metal music. Similar instruments usually generate similar spectral timbres, and similar rhythms will lead to similar temporal variation. As music signature describes temporal spectral characteristics of a local audio clip, it is expected that music pieces of a similar style will share some similar signatures, just like documents on similar topics usually share similar keywords. Thus, it is possible to make music recommendation practical by retrieving pieces with similar signatures. In other words, in our system, the criterion for recommendation is to find out music pieces with similar time-varying timbre characteristics.

In this section, we describe in detail the proposed idea of recommendation-by-search, and how it is implemented, based on the efficient music indexing scheme introduced in Section 3. In the following, we first introduce how to select signatures as query terms from a seed piece, and then present the criteria for designing the relevance-ranking function. Finally, we present the proposed strategy for automated playlist creation.

4.1 Music Snippet-based Query Selection

How to select proper signatures as query terms from a piece is not a trivial problem. First, not all the signatures in a piece are representative to its content. Second, too many query terms will drop the search performance significantly (on average, a piece around 5 minutes has more than 2,000 signatures). Actually, people like and remember one piece mostly because some short but impressive melody clips recur in the piece. Therefore, it is reasonable to select query terms only from such typical and repetitive segments, which have been called music *snippets* or *thumbnails*.

There have been several reported approaches for music snippet extraction [5, 20]. Here, we simply utilize a revised algorithm of that proposed in [5], as their approach was also based on audio signatures. In the implementation, we extract three snippets from the front, middle, and back parts of a piece [3], and each snippet is a segment of around $10 \sim 15$ seconds [5]. However, we still face the "long query" problem as there are still about 100 signatures in such a 15-second segment. It is still a large burden for the search engine.

Considering that music is a continuous stream and the two adjacent signatures have around 4-second overlaps, the L_2 distances between adjacent signatures are usually small, unless some distinct changes happen in the signal. Thus, such signatures can be further compacted by grouping signatures close enough to each for reducing the number of query terms. In our system, a bottom-up hierarchical clustering is performed on signatures from one snippet, and the clustering is stopped when the maximum distance between clusters is larger than $R/2$. For each cluster, the signature closest to the centre is reserved as a query term. Experimentally, the query terms can be reduced to 1/10 after the clustering. Finally, by combining adjacent signatures in a same cluster, a music snippet is converted to a query, which is represented with a sequence of (term, duration) pairs, as:

$$Q \sim [(q_1^Q, t_1^Q), \cdots, (q_i^Q, t_i^Q), \cdots, (q_{N_Q}^Q, t_{N_Q}^Q)], q_i^Q \in \mathcal{S}_Q \quad (7)$$

where q_i^Q and t_i^Q are the signature and the duration of the i^{th} term, $\mathcal{S}_Q = \{s_1, s_2, \cdots, s_{N_{UQ}}\}$ is the set of all the N_{UQ} unique terms in the query, and N_Q is the query length.

4.2 Rank Criteria

Relevance ranking is a crucial component of almost all search related problems. In text search, relevance ranking has been well studied such as the BM25 algorithm in [26]. The relevance ranking in music search has similar features with that in text search, but also has its own characteristics. As shown in (7), query terms here have *duration* information, and their *temporal order* is also important. Moreover, as the search is similarity-based but not identical matching, the *confidence* of such a matching should also be considered in the ranking.

First, let us look back the search process, and describe how the search results are obtained and organized for ranking. As introduced in Section 3.2, a query term (signature) will be hashed into L buckets with LSH, and the pieces indexed in these L buckets will be merged as a result list of this term. For a hit point (also a signature in a piece in the index), its similarity to the query term can be approximated by the number of buckets it belongs to over the whole L buckets (according to the LSH theory, the closer two signatures are, the higher probability they are in a same bucket). Such a similarity can be considered as a confidence of this matching. After going through all the unique terms in the query, their result lists are further combined to a candidate set [4] for relevance ranking. For each candidate piece in the set, its matching statistics can be represented with a triple sequence by merging adjacent hit points of a same term into a segment, as shown in Figure 4. A triple is in the form of (q^R, t^R, c^R), where q^R is the matched term, t^R is the segment duration, and c^R is the average matching confidence of the hit points in this segment.

In ranking, each candidate piece is further divided into fragments, as shown in Figure 4, if the time interval Δt between two matching segments is larger than a pre-defined

[3]There are usually several repetitive segments for a piece, and the snippet detection algorithm also returns multiple candidates. To cover more reasonable snippets, we experimentally select three most possible candidates from different parts of a piece.

[4]Here, it is assumed the search operation is "OR", as it cannot be expected all these terms in a query will exist in one another piece.

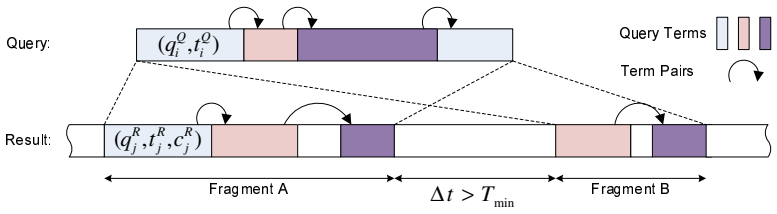

Figure 4: An example of the music search result matching. The candidate piece has two fragments (A and B) that possibly match the query.

threshold T_{min} (which is set as 15 seconds). Then, we compute the relevance scores for all the fragments, and the maximum is returned as the score of the candidate piece.

Considering of the characteristics of music search, in our approach, the relevance of a fragment is mainly based on the *matching ratio* and *temporal order*, and also integrates the *term weight* and the *matching confidence* above.

First, similar to the Robertson/Sparck weight [26] in text retrieval, the weight of the i^{th} term in S_Q is defined as:

$$w_i = \log \frac{V_0 - n_i + 0.5}{n_i + 0.5} \tag{8}$$

where V_0 is the total number of pieces in the dataset (i.e., the data scale defined in Section 3.2.2, and n_i is the length of the result list of the i^{th} term. The sum of all the term weights in S_Q is further normalized to one. In this way, we assign lower weights to popular terms while higher weights to special terms, just like the inverse document frequency (*idf*) utilized in text retrieval.

Then, our ranking function is defined as a linear combination of the measurements of the *matching ratio* f_{ratio} and the *temporal order* f_{order}, as:

$$f_{ranking} = f_{ratio} + f_{order} \tag{9}$$

In details:

- f_{ratio} is defined as:

$$f_{ratio} = \frac{1}{N_{UQ}} \sum_{i=1}^{N_{UQ}} \frac{\min(d_i^Q, d_i^R)}{\max(d_i^Q, d_i^R)} \cdot w_i \tag{10}$$

  where d_i^Q and d_i^R are the durations of the i^{th} term occurring in the query and in the fragment, respectively:

$$d_i^Q = \sum_{k|q_k^Q = s_i} t_k^Q; \quad d_i^R = \sum_{k|q_k^R = s_i} t_k^R \tag{11}$$

  In (10), the *matching ratio* and combined with the *term weight*.

- f_{order} is defined as:

$$f_{order} = \frac{1}{N_Q - 1} \sum_{i=1}^{N_Q - 1} P_{occur}(q_i^Q, q_{i+1}^Q) \tag{12}$$

  where $P_{occur}(q_i^Q, q_{i+1}^Q)$ is the maximum confidence of the pair (q_i^Q, q_{i+1}^Q) occurred in order in the result fragment, as:

$$P_{occur}(q_i^Q, q_{i+1}^Q) = \max_{j|q_j^R = q_i^Q \& q_{j+1}^R = q_{i+1}^Q} (c_j^R \cdot c_{j+1}^R) \tag{13}$$

  In (13), the *temporal order* and *matching confidence* are combined together.

In this way, fragments with larger matching ratio and more ordered term pairs are ranked with higher relevance scores, based on which corresponding candidate pieces are sorted for further recommendation.

4.3 Automated Playlist Creation

Up till now, we have implemented a search-based approach to find recommendations for a given piece from a music collection. In practice, users still need a continuous playlist, which can automatically expand with time. Here, we also present one strategy for automated playlist creation using the recommendation-by-search.

Various approaches have been proposed to automatically generate music playlists in previous works [4, 16, 23–25]. In general, a key idea is to provide an optimum compromise between the desire for repetition and the desire for surprise [22]. In another word, a good recommender should suggest both popular pieces with similar attributes ("stick to the seed"), and new pieces to provide fresh feeling ("drift for surprise"). However, for most content-based recommendation systems, finding novel songs becomes an unavoidable problem as their criterion is to find out similar pieces (while for CF-based recommendation, this issue can be well solved using the power of social community). Thus, our approach does have such a limitation in finding novel songs. To improve the diversity of recommendation, here, we heuristically add some dynamics when creating playlists. In details, the playlist is generated as following steps:

1. User manually assigns a piece as a seed.
2. Extract three snippets from the seed piece, as described in Section 4.1, to construct three queries for search. The first result of each query is added to the playlist. Thus, the suggestions are still acoustically similar with the seed. In this step, we intend to meet the requirement of "stick to the seed".
3. Randomly select a piece from the top three suggestions as a new seed, and go to step 2. Thus, the new seed is different with the original one, and will drive the playlist to a somewhat new style. In this step, we intend to meet the requirement of "drift for surprise".

User interactions can be easy integrated into the above process. For example, users can also tag any parts they are interested in a piece, and the playlist will be dynamically updated using queries generated from the tagged parts, instead of from the snippets by default.

5. EVALUATIONS AND DISCUSSIONS

In this section, we present results of the proposed recommendation-by-search, including the system efficiency, quantitative evaluations on both acoustic and genre consistencies, and sub-

jective evaluations from a user study. Based on these evaluations, we want to show that: 1) our approach is effective and efficient on various scales of music collections; and 2) the recommendation quality is also promising and is close to some state-of-the-art commercial systems.

5.1 Datasets and Experimental Settings

In experiments, we collected 114, 239 pieces (from 11, 716 albums) in mp3 and wma formats. To simulate music collections with different scales, we randomly sample some albums from all the 11, 716 albums, and construct four collections: $\mathbf{C}_1$ (1,083 pieces in 106 albums), $\mathbf{C}_2$ (5,126 pieces in 521 albums), $\mathbf{C}_3$ (9,931 pieces in 1007 albums), and $\mathbf{C}_4$ (all the pieces). These collection scales are selected to simulate the scenarios of recommendation on portable devices, personal PCs, and online radio services.

To evaluate the performances of the proposed solution on various scales of collections, for each collection, we created 20 playlists with the seed pieces listed in Table 2. For comparison, we also recorded the recommendation lists from a state-of-the-art online music recommendation service, Pandora [3], using the same 20 seeds. In addition, we still generated 20 playlists in shuffle model by randomly selecting pieces from the collections. The length of all the playlists is fixed to 10. Thus, in the experiment, we constructed 6 playlist collections, with 20 playlists in each.

As a side note, here we want to give more explanation to why we chose Pandora for comparison. Although there are some related techniques in the literature for automated and acoustic-based music recommendation, it is still hard to compare ours with them, since their implementation details and parameter settings are unavailable. Moreover, such comparisons are also unfair because the data collections used in various papers are different. This is why there are few such cross-system comparisons in music recommendation literature. In this paper, we try to situate the recommendation quality of our approach using two relatively fair references-random shuffle and Pandora. Pandora is public for access, and it is a top-ranked commercial recommendation service. It should be noted that we don't expect our automated and only acoustic-based system can exceed the performance of Pandora, as it leverages metadata and acoustic-related information, as well as many expert annotations. Thus, Pandora acts as a referee in the following evaluations. More discussions please refer to Section 5.5.

5.2 System Efficiency

In the experiment, we employ a PC with 3.2 GHz Intel Pentium 4 CPU and 1GB memory to evaluate the system efficiency.

We first evaluate the performance of the front-end, i.e., audio processing and music signature extraction. We randomly select 100 pieces in either mp3 or WMA format from the dataset. The average duration is 5.2 minutes. It took 3 minutes and 51 seconds for the front-end (including the steps of mp3/WMA decoding, down-sampling, MCLT, OPCA, and LSH-hashing) to parse all the 100 pieces. If the snippet extraction is also included, the total time cost is 5 minutes and 57 seconds. That is, 3.57 seconds are required on average to process a seed piece in recommendation. However, in most applications the seed piece is also a member of the music collection, and the snippets and query terms can be pre-generated and stored. The indexing time of the largest collection $\mathbf{C}_4$ is about 87 hours, and the detailed index size of each collection is listed in Table 3.

To evaluate the online search performance, for each collection, we carried out 1, 000 queries (with around 13.4 terms each) and the average performances are shown in Table 3. From Table 3, it is first observed that the memory costs of our system on various collections are relatively stable, and such memory cost is also acceptable for most desktop applications on PCs. Second, the average search time increases with the data scale, but is also acceptable for most applications. The search time here includes retrieving inverted indexes from ($\#\text{term} \times L$) hush buckets, merging, and ranking the search results. In $\mathbf{C}_1$, as most of the index can be cached in memory, the speed is quite fast. When index increases with the data scale, the search time becomes longer, as more disk I/O are needed for cache exchange. Actually, when the data scale is extremely large, the search operation can be easily distributed to multiple machines to accelerate the process time.

Another statistic shown in Table 3 is the average number of returned results. As discussed in Section 3.2, we need to assure enough results are returned for recommendation on various scale of collections. From Table 3, the resulting number can be roughly kept in the range of $500 \sim 1000$. More detailed, there are around 45% of pieces in $\mathbf{C}_1$ are returned for each query; while with $\mathbf{C}_4$ the percentage is only around 0.9%. However, the number of results is still increased with the data scale, as the LSH is designed to bind the worst conditions, while in real data the hitting probability is much higher than expected.

In general, it indicates our method of the scale-sensitive music indexing is effective in practice. In various music scales (application scenarios), our system can guarantee a return of a proper number of suggestions within an acceptable response time. In the following, we evaluate both the quantitative and subjective qualities of our recommendation results in Section 5.3 and Section 5.4.

5.3 Quantitative Evaluation

To the best of our knowledge, there is still not a sophisticated method to give a quantitative evaluation to music recommendation. In this work, we tried to utilize some indirect evidence for quantitative comparisons. One is the *acoustic consistency*, to verify the suggestions from the acoustic-level. The other is the *genre consistency*, to verify the suggestions from the metadata-level.

5.3.1 Acoustic Consistency

The acoustic consistency is to verify how close those suggested pieces are in the low-level acoustic space, and also has been utilized in some previous works for music recommendation [16, 19]. Here, we adopted a GMM-based approach [12, 16] to measure the distance between two pieces. In implementation, each piece in a playlist is modeled with a GMM in the $d = 64$ dimensional MCLT spectrum space (the same one we adopted in Section 3.1 for music signature extraction), as:

$$f(x) = \sum_{i=1}^{k} \alpha_i \mathcal{N}(x; \mu_i, \Sigma_i) = \sum_{i=1}^{k} \alpha_i f_i(x) \qquad (14)$$

where μ_i, Σ_i, and α_i are the mean, covariance, and weight of the i^{th} Gaussian component $f_i(x)$, respectively; and k is the number of mixtures (which is set as 10 experimentally). The distance between two GMMs $f(x)$ and $g(x)$ is then defined:

$$d(f, g) = \frac{1}{2}(\vec{d}(f, g) + \vec{d}(g, f)) \qquad (15)$$

Table 2: Information of the seed pieces in experiments.

No.	Track	Artist	Genre
1	Lemon Tree	Fool's Garden	Pop
2	My Heart Will Go On	Celine Dion	Pop
3	Candle in the Wind	Elton John	Pop
4	Soledad	Westlife	Pop
5	Say You, Say Me	Lionel Richie	Pop
6	Everytime	Britney Spears	Pop
7	As Long As You Love Me	Backstreet Boys	Pop
8	Right Here Waiting	Richard Marx	Rock
9	Yesterday Once More	Carpenters	Rock
10	It's My Life	Bon Jovi	Rock
11	Tears in Heaven	Eric Clapton	Rock
12	Take Me to Your Heart	Michael Learns to Rock	Rock
13	What'd I Say	Ray Charles	R&B
14	Beat It	Michael Jackson	R&B
15	Fight For Your Right	Beastie Boys	Rap
16	Does Fort Worth Ever Cross your Mind	George Strait	Country
17	Cross Road Blues	Robert Johnson	Blues
18	Born Slippy	Underworld	Electronic
19	Scarborough Fair	Sarah Brightman	Classical
20	So What	Miles Davis	Jazz

Table 3: The usages of disk, memory, and CPU on $\mathbf{C}_1 \sim \mathbf{C}_4$.

	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{C}_3$	$\mathbf{C}_4$
Index on Disk	70M	414M	787M	9.16G
Runtime Memory in Search	42.5M	43.3M	43.5M	47.1M
Average Search Time	0.27s	1.41s	1.72s	2.53s
Average Result Number	491	632	758	985

where $\vec{d}(f, g)$ is the direct distance from f to g:

$$\vec{d}(f,g) = \sum_{i=1}^{k} \alpha_i \min_{j,1\le j\le k} KL(f_i||g_i) \quad (16)$$

Here, the Kullback-Leibler (KL) divergence between two Gaussian components is defined as:

$$KL(\mathcal{N}(x;\mu_1,\Sigma_1)||\mathcal{N}(x;\mu_2,\Sigma_2)) =$$
$$\frac{1}{2}[\log\frac{|\Sigma_2|}{|\Sigma_1|} + tr(\Sigma_2^{-1}\Sigma_1) + (\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2) - d] \quad (17)$$

In this way, for each playlist we compute all the pair-wise distances between pieces. After going through all the 20 playlists in a collection, the distribution of such GMM-based distances on the collection is obtained and can be approximated by a Gamma distribution, similar to that introduced in Section 3.2.2. Figure 5 illustrates the approximate distance distributions on all the six playlist collections in our experiments.

From Figure 5, it is found that the average pair-wise distance in shuffle is the largest, while $\mathbf{C}_4$ is the smallest. This indicates that pieces suggested by our search-based approach still have similar acoustic characteristics in the track-level, although only signatures in snippet parts are used for search. This indicates our recommendation-by-search can satisfy the assumption of acoustic-based music recommendation. With the decrease of the data scale, from $\mathbf{C}_4 \sim \mathbf{C}_1$, the average distance becomes larger, as well as the deviation of the distribution. In Figure 5, the distribution of Pandora is in the middle of the shuffled approach and those generated using our approach. This indicates acoustic features may also

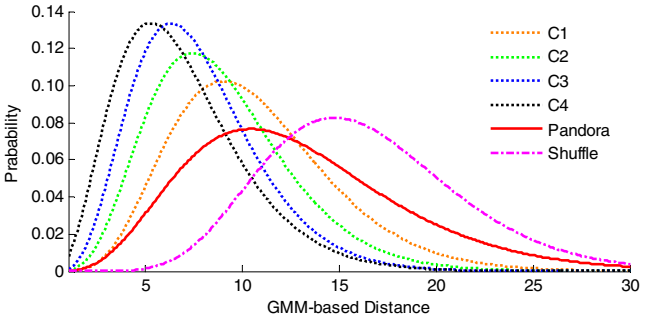

Figure 5: The distributions of the GMM-based pair-wise distance of pieces in a playlist, for the six playlist collections, respectively.

be considered in Pandora, but their recommendations are not only based on the acoustic attributes. This observation is consistent with the online introduction of Pandora [3], that is, it also leverages expert annotations such as culture and emotion to generate their playlists. Thus, in Pandora, pieces with similar annotations are also possibly selected for recommendation, although their low-level acoustic features may be quite different.

5.3.2 *Genre Consistency*

A music genre is a category of pieces of music that share a certain style, and is one of the basic tags in music industry. Although the genre classifications are sometimes arbitrary and controversial, it is still possible to note similarities be-

Table 4: Entropy of the genre distribution on the six playlist collections.

	Pandora	Shuffle	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{C}_3$	$\mathbf{C}_4$
Mean	0.23	0.56	0.32	0.40	0.38	0.35
Std	0.15	0.08	0.13	0.17	0.15	0.16

tween musical pieces, and thus is widely used in metadata-based music recommendation [4, 23–25]. To guarantee the genres used in the experiment are as accurate as possible, we utilized *AllMusic* (www.allmusic.com) – the most authoritative commercial music directory – to manually verify the genre of each piece. In total, nine basic genre categories: Pop, Rock, R&B, Rap, Country, Blues, Electronic, Classical, and Jazz, are adopted for classification.

The evaluation of genre consistency here is similar to that presented in [16]. That is, the Shannon entropy is utilized to measure the genre distribution of pieces in a playlist. The Shannon entropy is defined as:

$$H(x) = -\sum_{x} p(x) \log_{10} p(x) \tag{18}$$

where $p(x)$ is the percentage of a given genre in a playlist. Here, considering the length of a playlist is 10, we adopt $\log_{10}(\cdot)$ in (18) thus the entropy of the worst case (the 10 pieces in a playlist are from 10 different genres) is 1. And for the ideal case (all the 10 pieces are from a same genre), the entropy is 0. The statistics of the entropies on the 6 collections are listed in Table 4.

There is not an authoritative criterion to describe what the genre distribution should be like for an ideal playlist [16]. Here, by comparing the average entropies of playlists from Pandora and in shuffle model, we just assume the lower the entropy, the better the playlist quality. In Table 4, the entropy of playlists in shuffle is the highest and with small deviation, and it indeed should be close to the genre distribution of the whole music collection. The genre entropies of the playlists from $\mathbf{C}_1 \sim \mathbf{C}_4$ are around 0.3∼0.4, and are between Pandora and the shuffle one. As genre is actually one of the criteria utilized for recommendation in Pandora [3], the distribution on Pandora is the most concentrated. Through the comparison, it indicates that our approach can still keep the genre consistency, to a certain extent.

5.4 Subjective Evaluation

To evaluate the performance in practice, we also conducted a small user study by inviting 10 college students as testers. Considering the work load, we randomly selected 5 playlists from each collection for each tester. Thus, each tester evaluated 30 playlists through listening to them one by one, and the collection information was blind to the testers. The testers were asked to assign a rating ranging score from 1 to 5 to each playlist. The rating criteria are: 1 ("totally unacceptable"), 2 ("marginally acceptable, but still inconsistent"), 3 ("acceptable, and basically consistent"), 4 ("acceptable, with some good suggestions"), and 5 ("almost all good suggestions"). Here, "acceptable" is defined as "it is OK to finish the playlist without interruption". To remove the individual bias, ratings from each tester are first re-normalized before analysis. Then, the normalized ratings from various testers are averaged on each playlist collection, and the corresponding mean and standard deviation are kept for comparison, as shown in Table 5.

Table 5: Statistics of the subjective ratings for the six playlist collections.

	Pandora	Shuffle	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{C}_3$	$\mathbf{C}_4$
Mean	4.29	1.73	3.81	3.85	3.88	3.87
Std	0.69	0.52	0.91	0.97	0.95	0.96

From Table 5, it is clear that the highest subjective rating is achieved on Pandora, with an average rating close to 4.3. The ratings from $\mathbf{C}_1 \sim \mathbf{C}_4$ are around 3.85, which indicates that with our approach, the suggestion qualities are still acceptable and suffer little from the data scales, especially when the scales are large enough (such as $\mathbf{C}_3$ and $\mathbf{C}_4$). The performance of the playlists in shuffle is the worst, their average ranking is lower than 2. However, an interesting phenomenon is that the standard deviation on the shuffle collection is the smallest, which suggests subjective judgments are more consistent using it. Similarly, the subjects also show consistent satisfaction for Pandora. While in comparison, such deviations of $\mathbf{C}_1 \sim \mathbf{C}_4$ are notably higher. It indicates that the suggestion qualities of our approach are not as stable as that of Pandora, and need to be improved in the future work.

5.5 Discussion

The above evaluations have shown our solution can achieve encouraging and stable performance on various scales of music collections, and is efficient in practice. The general performance is much better than that in shuffle, and is close to Pandora. However, it still needs improvement in comparison with Pandora. Pandora was created by the Music Genome project [3], which is trying to "*create the most comprehensive analysis of music ever*". In Music Genome, a group of musicians and music-loving technologists were invited to carefully listen to pieces and label "*everything from melody, harmony and rhythm, to instrumentation, orchestration, arrangement, lyrics, and of course the rich world of singing and vocal harmony*". Thus, the recommendation of Pandora has integrated both meta- and acoustic- information, as well as human knowledge from music experts. That is why it achieved the best subjective satisfaction in the experiments. However, Pandora requires a great amount of manual/expert labeling works, which is expensive and is not available without great difficulty in many applications, such as music recommendations on personal PCs or portable devices.

In comparison, our solution can be conveniently deployed to both desktop and web services. Especially for desktop-based applications, our approach can be naturally integrated into desktop search, to facilitate users search, browsing, and discovery local personal music resource. Furthermore, if metadata and user listening preferences are available, our solution can be further improved by combining local acoustic-based search results with CF-based and meta-based information retrieved from the Web. This is one of our directions in the future.

6. CONCLUSIONS

In this paper we have presented a search-based solution for scalable music recommendation. In this solution, through acoustic analysis, music pieces are first transformed to sequences of music signatures. Based on this, an LSH-based scale-sensitive method is presented to index the music pieces

for effective similarity search. According to a given data scale, this method can numerically estimate the appropriate parameters to index various scales of music collections, and thus guarantees that an optimum number of nearest neighbors can be returned in search. In the recommendation stage, representative signatures from the snippets of a seed piece are first selected as query terms to retrieve pieces with similar melodies from the indexed dataset. Then, a relevance function is designed to sort the search results by considering criteria like *matching ratio*, *temporal order*, *term weight*, and *matching confidence*. In addition, we also propose a strategy to generate dynamic playlists using the search results. Experimental evaluations have shown that the proposed approach achieved promising performance on system efficiency, content consistency, and subjective satisfaction for various music collections from around 1,000 music pieces to more than 100,000 pieces.

Although the proposed solution has been verified to be feasible for music recommendations, there is still considerable room for further investigation and improvement. For example, besides the relevance (dynamic) ranking, static ranks such as sound quality and music popularity can also be integrated to find better suggestions. Moreover, more sophisticated acoustic features should be evaluated to discover those that are more suitable for music recommendation. Lasting personalized music recommendation, user preferences should be modeled by tracking operational behavior and listening histories. These are directions of our future work.

7. REFERENCES

[1] Last.fm – The Social Music Revolution. http://www.last.fm.

[2] Owl multimedia ≫ use YOUR music to find New music! http://www.owlmm.com.

[3] Pandora Internet Radio and Music Genome Project. http://www.pandora.com.

[4] J.-J. Aucouturier and F. Pachet. Scaling up music playlist generation. In *Proc. IEEE ICME'02*, volume 1, pages 105–108, Lausanne, Aug. 2002.

[5] C. J. C. Burges, D. Plastina, J. C. Platt, E. Renshaw, and H. S. Malvar. Using audio fingerprinting for duplicate and thumbnail generation. In *Proc. IEEE ICASSP'05*, volume 3, pages 9–12, Philadelphia, PA, USA, Mar. 2005.

[6] C. J. C. Burges, J. C. Platt, and S. Jana. Distortion discriminant analysis for audio fingerprinting. *IEEE Trans. Speech and Audio Processing*, 11(3):165–174, May 2003.

[7] P. Cano, E. Batlle, E. Gómez, L. Gomes, and M. Bonnet. *Audio fingerprinting: concepts and applications*, chapter 17, pages 233–245. Computational Intelligence for Modelling and Prediction. Springer-Verlag, 2005.

[8] P. Cano, M. Markus, and N. Wack. An industrial-strength content-based music recommendation system. In *Proc. SIGIR'05*, pages 673–673, Salvador, Brazil, Aug. 2005.

[9] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *Proc. PODS'98*, pages 59–68, Seattle, USA, Jun. 1998.

[10] W. Cohena and W. Fana. Web-collaborative filtering: recommending music by crawling the Web. *Computer Networks*, 33(1-6):685–698, Jun. 2000.

[11] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. SCG'04*, pages 253–262, Brooklyn, NY, USA, Jun. 2004.

[12] J. Goldberger and S. Roweis. Hierarchical clustering of a mixture model. In *Proc. NIPS'04*, pages 505–512, Vancouver, Canada, Dec. 2004.

[13] Y.-C. Huang and S.-K. Jeng. An audio recommendation system based on audio signature description scheme in MPEG-7 audio. In *Proc. IEEE ICME'04*, pages 639–642, Taipei, Taiwan, Aug. 2004.

[14] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. SOTC'98*, pages 604–613, Dallas, Texas, USA, Jun. 1998.

[15] D. Jiang and G. Xu. Tunable locality sensitive hashing: a unified approach to near duplicate detection. Submit to ICDE'08, Cancún, México, Apr. 2008.

[16] P. Knees, T. Pohle, M. Schedl, and G. Widmer. Combining audio-based similarity with Web-based data to accelerate automatic music playlist generation. In *Proc. ACM MIR'06*, pages 147–153, Santa Barbara, CA, USA, Oct. 2006.

[17] N. Kosugi, Y. Nishihara, T. Sakata, M. Yamamuro, and K. Kushima. A practical query-by-humming system for a large music database. In *Proc. ACM Multimedia'00*, pages 333–342, Los Angeles, Oct. 2000.

[18] Q. Li, B. M. Kim, D. H. Guan, and D. Oh. A music recommender based on audio features. In *Proc. SIGIR'04*, pages 532–533, Sheffield, Jul. 2004.

[19] B. Logan. Music recommendation from song sets. In *Proc. ISMIR'04*, pages 425–428, Barcelona, Oct. 2004.

[20] L. Lu and H.-J. Zhang. Automated extraction of music snippet. In *Proc. ACM Multimedia'03*, pages 140–147, Berkeley, USA, Oct. 2003.

[21] H. S. Malvar. Fast algorithm for the modulated complex lapped transform. *IEEE Signal Processing Letters*, 10(1):8–10, Jan. 2003.

[22] E. Pampalk, T. Pohle, and G. Widmer. Dynamic playlist generation based on skipping behavior. In *Proc. ISMIR'05*, pages 634–637, London, Sept. 2005.

[23] S. Pauws, W. Verhaegh, and M. Vossen. Fast generation of optimal music playlist using local search. In *Proc. ISMIR'06*, Victoria, Canada, Oct. 2006.

[24] J. C. Platt, C. J. C. Burges, S. Swenson, C. Weare, and A. Zheng. Learning a Gaussian process prior for automatically generating music playlists. In *Proc. NIPS'01*, pages 1425–1432, Vancouver, Canada, Dec. 2001.

[25] R. Ragno, C. J. C. Burges, and C. Herley. Inferring similarity between music objects with application to playlist generation. In *Proc. ACM MIR'05*, pages 73–80, Singapore, Nov. 2005.

[26] S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 extension to multiple weighted fields. In *Proc. CIKM'04*, pages 42–49, Washington, D.C., USA, Nov. 2004.

[27] K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. Okuno. Hybrid collaborative and content-based music recommendation using probabilistic model with latent user preference. In *Proc. ISMIR'06*, Victoria, Canada, Oct. 2006.