Song Recommendation for Social Singing Community

Kuang Mao[1] Ju Fan[2] Lidan Shou[1] Gang Chen[1] Mohan Kankanhalli[2]
[1]College of Computer Science, Zhejiang University, Hangzhou, China
[2]School of Computing, National University of Singapore, Singapore
mbill@zju.edu.cn; fanj@comp.nus.edu.sg; {should,cg}@zju.edu.cn; mohan@comp.nus.edu.sg

ABSTRACT

Nowadays, an increasing number of singing enthusiasts upload their cover songs and share their performances in online social singing communities. They can also listen and rate other users' song renderings. An important feature of the social singing communities is to recommend appropriate singing-songs which users are able to perform excellently.

In this paper, we propose a singing-song recommendation framework to make song recommendation in social singing community. Instead of recommending songs that people like to listen, we recommend suitable songs that people can sing well. We propose to discover the song difficulty orderings from the song performance ratings of each user. We transform the difficulty orderings into a difficulty graph and propose an iterative inference algorithm to make singing-song recommendation on the difficulty graph. The experimental result shows the effectiveness of our proposed framework. To the best of our knowledge, our work is the first study of singing-song recommendation in social singing communities.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering; H.5.5 [**Sound and Music Computing**]: Systems

General Terms

Algorithms; Experimentation

Keywords

Singing-song recommendation; social singing community; recommender systems

1. INTRODUCTION

Singing and listening in Karaoke is an enjoyable form of entertainment for many people. With the popularity of high-speed wireless connectivity, the Web today is increasingly seeing a new form of online Karaoke known as the *Social Singing Community* (SSC). A singer in a SSC can record a performance (or album) via her/his device, and then publish to SSC, allowing for access to it at any place and any time. On the other hand, a listener, after enjoying the performances disseminated by the system, gives ratings, likes, or comments to the performances or singers. To date, a great number of SSCs have emerged on the Web. Some popular ones include *Smule*[1], *5sing*[2], and *Sing!*[3]. For example, 5sing has a population of 1.3 million active users and a repository of 7.5 million cover songs sung by its users by 2013. Figure 1 shows the interface of a well-known mobile SSC application called Sing!.

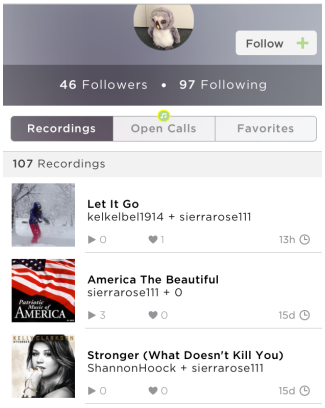


Figure 1: Screenshot of Sing! interface

While SSC can serve as a conventional platform of listening to songs for the listeners, it provides an essential "virtual stage" for the amateur singers. Unlike conventional music communities in which listeners rate the songs and interact with other listeners, SSC focuses more on the *singers*, by encouraging interaction between the singers and the listeners via ratings on the performances.

One important service in SSC is to provide singing-song recommendation for singers. Despite the abundance of studies in song recommendation [7], the problem of singing-song recommendation in SSC has its unique characteristic which has not been addressed adequately in the literature. Specifically, the conventional song recommendation focuses on discovering songs that a user likes to listen. In contrast, singing-song recommendation aims to find the songs which are easier to lead to high *singing performances*. In particular, higher singing performance in SSC means the corre-

[1]http://www.smule.com/
[2]http://www.5sing.com/
[3]https://itunes.apple.com/app/sing!-karaoke-by-smule/id509993510?mt=8

MM '14, November 3-7,2014, Orlando, USA

http://dx.doi.org/10.1145/2647868.2654921.

sponding songs can help a singer obtain more appreciation and higher ratings from the listeners. Acquiring higher rating is very important to the singer, because it can not only bring acknowledgement and encouragement, but also help her promote the influence in the community and further attract more fans. In the rest of the paper, for ease of presentation, we use "song s_1 is easier than s_2" to represent that s_1 is easier to incur higher singing performance than s_2. Note that this may not necessarily reflect the difficulty in terms of acoustic features (melody, pitch range, etc.): some easy-to-sing songs may not bring impressive performances and high ratings due to their unattractive melodies.

Although a previous study published in [15] has considered singing-song recommendation, it cannot address our problem in SSC. The work in [15] proposed a solution called competence based song recommendation, which relies on complex modeling of singer's vocal competence and song profiles, and conducts a matching between singer competence and songs using a machine learning algorithm. Unfortunately, this approach is impractical for singing-song recommendation in SSC, due to the following reasons: First, the vocal competence modeling in [15] employs an expensive recording process which requires the user to sing every pitch in his/her pitch range from the softest to the loudest, under the supervision of music professionals. This requirement cannot be practically achieved in SSC. Second, the song profiles are acquired from music scores and manually annotated singing intensity data. Such process requires prohibitive manual works of cleaning, calibration, and annotation, which is impossible under the circumstance of large-scale SSC.

In this paper, we report our research on *SInging-song Recommendation* (SIR) for a social singing community. The main objective is to study how to recommend songs that users have large potentials to perform well and finally obtain high ratings from the listeners. Specifically, we develop a SIR framework which takes a user's singing history as input, and, from the singing history, predicts which songs the user can perform well. Our song recommendation is different from the traditional song retrieval which focuses only on matching users' listening interests. We pay particular attention to matching the users' singing performance needs. It also differs with Shou [15]'s work which focuses on matching songs with singer's singing model. We discover the songs' difficulty evidences in terms of performance from listener ratings and recommend songs that are most probable to incur better performance than the songs sung by a user in the history. To the best of our knowledge, this is the first work to deal with the SIR in a SSC.

Our singing-song recommendation faces two main technical challenges. The first one is how to determine whether a song is easier than another one, i.e., the difficulty relation in performance between two songs. As such knowledge of difficulty relation is not off-the-shelf, one approach is to ask a group of singers to judge the difficulty in performance between two songs. However, it is costly to employ people to compare the difficulty. Another approach is to indirectly discover the difficulty comparison results from SSC. The problem is we need to determine how reliable the obtained difficulty relations are.

The second challenge is how to make recommendation based on the aforementioned song difficulty relations. To make full use of these relations, we need to model these song difficulty relations so that we can infer new difficulty relations from existing ones. For song recommendation, it is challenging to estimate whether a user can perform a specific song well based on the inference from their singing history.

To solve the first challenge, we propose to discover the *song difficulty ordering* from the singing performance rating of the songs in users' singing history. We derive song difficulty orderings from the performance ratings given to the songs sung by the same user. This is not easy, however, as the diversity in singer's vocal competence and listener's preference may probably lead to inconsistent ratings. Such uncertainty poses the following questions to us: Is the discovered difficulty ordering valid for every singer? Are there "common" orders which are valid for most singers and ad-hoc ones which depend largely on individual singers. To answer these questions, we need to measure the *reliability* of song difficulty orderings as the result of derivation from human judgments.

To address the second challenge, we capture all the song difficulty orderings in a *difficulty graph*, where each vertex represents a song. We evaluate whether the user can sing a song well, using a quantity called *performance degree*. Given a song (vertex), this quantity can be estimated from its neighboring vertices in the graph, using a probabilistic inference technique. Since a user's singing history covers only a few vertices in the difficulty graph, we propose an *Iterative Probabilistic Inference (IPI)* algorithm in the graph which iteratively "spreads" the performance degrees from the user's singing history to other songs. Note that, the proposed recommendation framework can also be generalized to other competence-related recommendation problems such as books, games, where we only need to determine the difficulty relations between vertices in the difficulty graph.

The advantage of this approach in the current problem is that, the difficulty orderings discovered in one SSC can be reused and updated in another SSC system which is a way to ease the cold start problem in recommendation system. Given sufficient resources, it is also possible to employ music experts to build an ever-expanding song difficulty ordering database and then provide singing-song recommendation services for any social singing community. In this paper, we publish the very first difficulty ordering database of the songs used in our experiments[4].

Our contributions can be summarized as follows:

(1) We propose a singing song recommendation framework to solve the song recommendation problem for a social singing community.

(2) We propose a method for discovering the song difficulty orderings from the social singing community and measure the reliability of the discovered difficulty orderings.

(3) We present a difficulty graph to model song difficulty orderings and build up a probabilistic inference technique to estimate a song's performance degree.

(4) We introduce a ranking algorithm for singing song recommendation from a difficulty graph.

(5) Our experiment shows promising results for the proposed recommendation framework.

The rest of the paper is organized as follows: Section 2 introduces the related work. In Section 3, we define our research problem and provide an overview of the framework. Section 4 describes how to discover the song difficulty orderings from social singing community. Section 5 introduces

[4]http://db.zju.edu.cn/dataDownload/index.html

the inference technique in difficulty graph and describes the song recommendation algorithm. Section 6 describes the details of the experiments and the results. Finally, in Section 7 we conclude the paper.

2. RELATED WORK

In this paper, we study the problem of singing song recommendation for social singing community. This research is related to the traditional song recommendation, singing song recommendation and graph ranking algorithms. We will give a brief review of these works and explain the differences with our work.

2.1 Traditional Song Recommendation

Traditional song recommendation systems are designed for discovering songs that satisfy user's listening interests. The early methods for song recommendation such as [10] explore techniques in the domain of content based song recommendation. These techniques use low level features to represent user's preferences of the songs such as moods and rhythms and recommend songs that have a large similarity on low level features. However, it suffers from the semantic gap between low level features and user's preferences.

The classic way to recommend songs is via Collaborative Filtering (CF) [8], which can be divided into user-based CF [6] and item-based CF [14]. The user-based CF first predicts the human ratings for unrated items (songs) from a group of similar users and makes recommendation by ranking the predicted rating of each items. The intuition of CF (user-based) comes from the assumption that if users A and B have a strong similarity in the choice of listening songs, their songs can be recommended to each other. However, it is not always valid for recommending songs for singing. People will tend to sing some popular songs which are liked by many people, however they will also sing some unique songs. It is inappropriate to recommend such unique songs to others without considering the songs' difficulty in performance.

Another straightforward way is to use an ordinary graph to do song recommendation, it first builds a user-item bigraph and performs random walk to get the song ranking where those songs that share many connected users with the query song will get higher ranks. It intuitively recommends popular songs to satisfy user's interest needs. It also ignores taking song's difficulty in performance into consideration. Wu [16] proposed a next-song recommendation strategy for online karaoke users using personalized markov embedding. They also focus on matching user's singing interests, while ignoring singer's demands for high performance in karaoke.

In our proposed approach, we will discover song's difficulty in performance. The recommendation is based on the song difficulty orderings which guarantees that recommended songs are easier than the user's past sung songs.

2.2 Singing Song Recommendation

Singing song recommendation is a relatively new research area which was first proposed by Mao *et al.* [13] in 2012. Then Shou *et al.* [15] presented a competence based song recommendation system which utilizes a singer's digitized voice recording to recommend a list of songs based on the analysis of singer's vocal competence. They require a professional recording process to acquire user's vocal competence, called singer profile, and model the song profile which is the singing notes distribution within each song. After that, they propose a learning to rank scheme for recommending songs using the singer profile. Their system has two major drawbacks for recommending songs for a social singing community. The first one is that it requires a complex user's vocal competence acquisition process which is not fit for the social singing community. Another drawback is that it needs each song's score to build the song model which is hard to acquire for large number of songs in the social singing community setting.

In our work, we avoid modeling user's vocal competence and do not need to know song's score for singing song recommendation. We only need the users' singing history to do the recommendation.

2.3 Ranking on the Graph

Graph ranking studies the ranking method for the graph vertices. Most methods [4, 5, 18] are based on regularization theory for graphs. Zhou *et al.* [19] propose a universal ranking algorithm, which can exploit the intrinsic manifold structure of the data. They first build a weighted graph, define and normalize the weight of the edge. During each iteration, each vertex spreads its rank to the neighbors via the weighted network until the ranks converge. Agarwal [1] develops an algorithmic framework for learning ranking functions on both directed and undirected graph data. They build the graph which acts as a regularizer to make sure closely connected document will get a similar value. Baluja [3] proposed a video co-view graph and developed a random walk algorithm to do video recommendation.

For our graph ranking method, the basic idea is similar to that of Zhou *et al.* [3, 19]. We use a probabilistic modeling technique to model our difficulty graph. We rank the songs by iteratively doing probabilistic inference in the graph.

3. SINGING-SONG RECOMMENDATION

In this section, we formalize the problem of song recommendation for social singing community in Section 3.1 and introduce our recommendation framework in Section 3.2.

3.1 Problem Statement

Social Singing Community. Our work studies a social singing community with a set of users $U = \{u_1, u_2, \ldots, u_m\}$ and a collection of songs $S = \{s_1, s_2, \ldots, s_n\}$. In the community, each user $u_k \in U$ has a singing history that consists of a subset of songs in S, denoted by $S_k \subseteq S$. The user u_k could post this history in the community to let other users listen to the songs in S_k. Then, the listeners may evaluate u_k's performance on these songs by providing ratings on a scale from 1 to 5. We compute the average rating of listeners on each song $s_i^k \in S_k$ performed by u_k and denote this rating as r_i^k. The input phase of Figure 2 provides an example of user's singing histories. In this example, the user u_1 has a history consisting of songs s_1, s_2, etc., where the songs are associated with the average rating by listeners, e.g., rating for s_1 is 4.3 for user u_1.

Singing-song Recommendation. In this work, we aim to recommend the songs that users have the potential to perform well. To better illustrate this recommendation objective, let us consider a user u_k and two songs s_i and s_j in the song set $S - S_k$. If the performance of u_k on s_i could attract more attention from the listeners and the listeners would give higher ratings to this song, then song s_i would be preferable than s_j in our recommendation.

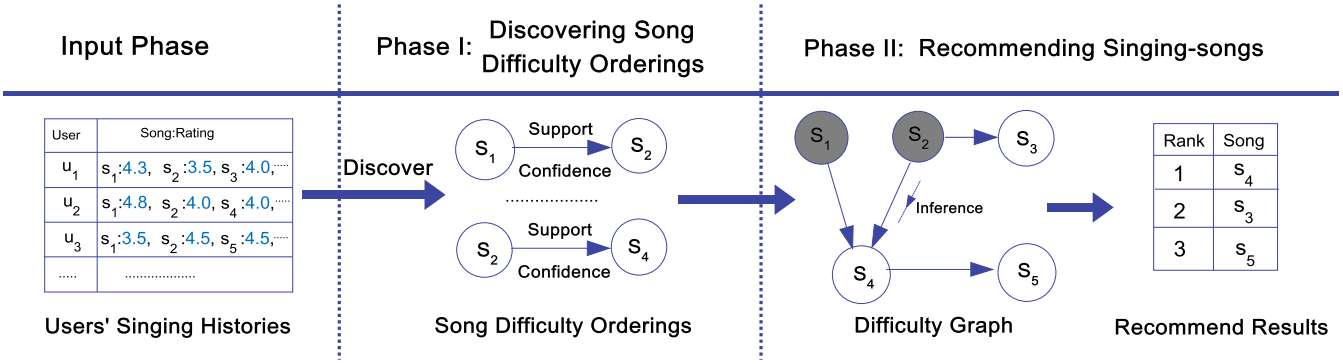


Figure 2: Overview of the singing-song recommendation framework

3.2 Recommendation Framework

Our framework for singing-song recommendation is illustrated in Figure 2. The framework takes as input the users' singing histories in the community and recommends songs to each individual user. As mentioned in the introduction, the basic idea is to infer whether a user can perform well on a song by considering song rendition difficulty. Intuitively, if a song is much more difficult than the singing history of a user, then the song is more likely to be inappropriate for the user, and it would not be recommended. Specifically, the framework recommends songs through two phases.

Phase I - Discovering song difficulty orderings. In this phase, we discover the ordering of any two songs in terms of *difficulty*, which captures whether song s_i is more difficult to perform well than s_j to the users, denoted by $s_i > s_j$. We discover the difficulty orderings from users' singing histories: if s_i has lower ratings than s_j in many singing histories, we conjecture that s_i is more difficult than s_j. Then, we evaluate the reliability of the conjecture by considering the following two factors.

1) The first factor is the *support* of the ordering, that is, how many singers support s_i is more difficult than s_j.

2) The second factor is the *confidence* of the ordering, that is, whether there are many other singing histories support the opposite conclusion, i.e., s_j is more difficult than s_i.

We will formally define song difficulty ordering and present our method of estimating support and confidence of difficulty ordering in Section 4.

Phase II - Recommending singing-songs. In this phase, we recommend songs that users have the potential to perform well. To this end, given a user u_k, we need to infer which songs are easier to get a good performance compared to the songs in u_k's singing history. We make the inference with the help of the difficulty orderings obtained in the first phase. More specifically, we first construct a difficulty graph based on difficulty orderings, where each vertex represents a song and a directed edge captures the difficulty ordering of two songs. Then, we develop an iterative probabilistic inference algorithm to obtain songs which are most likely to be "easier" to perform well for a user than those in her/his singing history. We will discuss the details of the algorithm in Section 5.

We illustrate our framework by using the example in Figure 2. In the first phase, the framework discovers difficulty orderings for song pairs and evaluates the reliability of each ordering. For instance, there are 2 users' singing history supporting $s_1 > s_2$ while only 1 singing history is supporting $s_2 > s_1$. Based on this, we estimate the support and confidence of $s_1 > s_2$ for evaluating the reliability of this ordering. In the second phase, we organize the songs into a graph based on difficulty orderings as shown in the right part of Figure 2. The gray vertices represent the songs in a user's singing history while the white vertices are the candidates to be recommended. Then, our inference algorithm applies an iterative strategy to infer the likelihood that each white vertex is easier than gray vertices and recommends the top songs with the highest likelihood scores.

4. SONG DIFFICULTY ORDERING

This section presents the technique of discovering difficulty ordering of singing-songs. We formalize the notion of song difficulty ordering in Section 4.1 and estimate the reliability of song difficulty orderings in Section 4.2. Finally, we discuss how to discover difficulty orderings from users' singing histories in Section 4.3.

4.1 Song Difficulty Ordering

The knowledge of users' performance on some songs can help us infer their performance on other songs. Intuitively, if a user performs well on a song s_i, then we can conjecture that she is also be capable of singing the songs that are "easier" to perform well than s_i. To formalize this, we introduce the idea of *song difficulty ordering* to capture whether a song is more difficult to perform well than another one. Specifically, given two songs s_i and s_j, we use the notation $s_i > s_j$ to represent one difficulty ordering, which means s_i is more difficult to perform well than s_j.

We introduce a *voting-based* strategy to determine the difficulty ordering of two songs. The basic idea is as follows. Suppose that a group of singers sing and vote which song from s_1 and s_2 is more difficult to get a good performance. Then, we can obtain a set of votes for difficulty comparison of s_1 and s_2. If more singers vote s_i as the difficult one, we can obtain a difficulty ordering $s_i > s_j$ based on these votes. We will discuss how we obtain singers' votes later in Section 4.3. More formally, we first introduce the difficult voting as follows.

Definition 1 (Difficulty Voting). *Difficulty voting is the process of voting the relative difficulty of two songs s_i and s_j in getting a better performance, where n_{ij} and n_{ji} represent the numbers of votes that support $s_i > s_j$ and $s_j > s_i$ respectively.*

For instance, if there are 7 singers think s_i is more difficult than s_j while 3 singers vote for s_j, then we have $n_{ij} = 7$ and $n_{ji} = 3$ in the difficulty voting of the two songs.

Now we define the song difficulty ordering of two songs.

Definition 2 (Song Difficulty Ordering). *The song difficulty ordering of song s_i and s_j is the majority choice in difficulty voting for $s_i > s_j$ and $s_j > s_i$.*

Continuing the above example, because s_i and s_j's difficulty voting results is $n_{ij} > n_{ji}$, we get the song difficulty ordering for the two songs as $s_i > s_j$.

4.2 Reliability of Difficulty Ordering

As singers may not always have agreement on the difficulty ordering, we need to estimate the reliability of difficulty ordering from the corresponding difficulty voting. We consider two constituents of reliability, namely the *support* and the *confidence* of difficulty ordering.

4.2.1 Support

We first consider the factor that how many singers support the ordering $s_i > s_j$. We denote the *Support* of difficulty ordering by $\texttt{supp}(s_i > s_j)$. Intuitively, a difficulty ordering which is supported by more singers would be more reliable than the one with less support. We define the support of the difficulty ordering

Definition 3 (Support of Difficulty Ordering). *The support of the difficulty ordering $s_i > s_j$, denoted as $\texttt{supp}(s_i > s_j)$, is the number of votes that $s_i > s_j$ gets.*

Then we need to measure whether a difficulty ordering gets sufficient supports. The difficulty orderings are used for recommendation. s_i will form difficulty orderings with many other songs, for example $s_i > s_t$. It would be more reliable to use the one with more supports to do recommendation. So we estimate the probability that $s_i > s_j$ gets sufficient supports as follow:

$$P_{suff}(s_i > s_j) = \frac{\texttt{supp}(s_i > s_j)}{\sum_{s_t \in S_\sigma} \texttt{supp}(s_i > s_t)} \tag{1}$$

where S_σ is a song set and each s_t in S_σ forms a song difficulty ordering $s_i > s_t$.

For example, if we have only two difficulty orderings $s_i > s_j$ and $s_i > s_t$ who have support of 3 and 2 respectively. Then $P_{suff}(s_i > s_j) = 3/(2+3)$.

4.2.2 Confidence

While considering how many singers support an ordering, the *support* does not take into account how many singers oppose that ordering. For example, the vote number of $s_i > s_j$ and $s_j > s_i$ is 10 and 9 respectively, although $s_i > s_j$ has many votes, the reliability of $s_i > s_j$ is doubtful due to the large number of opposing votes. To address this problem, we further consider how confidently can we "trust" a difficulty ordering as the *Confidence* of the difficulty ordering, denoted by $\texttt{conf}(s_i > s_j)$.

We define the confidence of difficulty ordering

Definition 4 (Confidence of Difficulty Ordering). *The confidence of the difficulty ordering $s_i > s_j$, denoted as $\texttt{conf}(s_i > s_j)$, is the probability of getting the judgment s_i is more difficult than s_j, given $s_i > s_j$ and $s_j > s_i$'s vote number n_{ij} and n_{ji}.*

As the definition, we represent $\texttt{conf}(s_i > s_j)$ in the probability form $P(s_i > s_j | n_{ij}, n_{ji})$. To compute $P(s_i > s_j | n_{ij}, n_{ji})$, we first use Bayes's theorem:

$$P(s_i > s_j | n_{ij}, n_{ji}) = \frac{P(n_{ij}, n_{ji} | s_i > s_j) P(s_i > s_j)}{P(n_{ij}, n_{ji})} \tag{2}$$

We assume s_i,s_j have equal probability to be more difficult which makes $P(s_i > s_j)$ and $P(s_j > s_i)$ the same value. Then

$$\frac{P(s_i > s_j | n_{ij}, n_{ji})}{P(s_j > s_i | n_{ji}, n_{ij})} = \frac{P(n_{ij}, n_{ji} | s_i > s_j)}{P(n_{ji}, n_{ij} | s_j > s_i)} \tag{3}$$

If we assume $P(s_i > s_j | n_{ij}, n_{ji}) + P(s_j > s_i | n_{ji}, n_{ij}) = 1$, together with Equation 3, we get

$$P(s_i > s_j | n_{ij}, n_{ji}) = \frac{P(n_{ij}, n_{ji} | s_i > s_j)}{P(n_{ij}, n_{ji} | s_i > s_j) + P(n_{ji}, n_{ij} | s_j > s_i)} \tag{4}$$

Then we need to calculate the probability of the voting result given the difficulty ordering. The subjects can only vote one of s_i and s_j. We consider the process of subject's voting as a Bernoulli trial. We set the human's voting accuracy to be p. The probability of observing n_{ij},n_{ji} is the Bernoulli distribution probability mass function:

$$P(n_{ij}, n_{ji} | s_i > s_j) = C^{n_{ij}}_{n_{ij}+n_{ji}} p^{n_{ij}} (1-p)^{n_{ji}} \tag{5}$$

$$P(n_{ji}, n_{ij} | s_j > s_i) = C^{n_{ji}}_{n_{ij}+n_{ji}} p^{n_{ji}} (1-p)^{n_{ij}} \tag{6}$$

Now we begin to estimate p from user's difficulty voting. We denote τ as our song difficulty ordering set. γ as an opposite difficulty ordering set with τ, their sizes being the same. For example, if $s_i > s_j \in \tau$, then $s_j > s_i \in \gamma$. We estimate p as follows

$$p = \frac{\sum_{(s_i > s_j) \in \tau} V(s_i > s_j)}{\sum_{(s_j > s_i) \in \gamma} V(s_j > s_i) + \sum_{(s_i > s_j) \in \tau} V(s_i > s_j)} \tag{7}$$

Where $V(s_i > s_j)$ is the difficulty votes $s_i > s_j$ gets.

Knowing $P(n_{ij}, n_{ji} | s_i > s_j)$ and $P(n_{ji}, n_{ij} | s_j > s_i)$, using Equation 4 we can obtain $P(s_i > s_j | n_{ij}, n_{ji})$

Similar to the sufficient of the support, we need to measure the probability of the difficulty ordering to be used in recommendation. We estimate whether the probability that $s_i > s_j$ gets a high confidence as follow:

$$P_{conf}(s_i > s_j) = \frac{\texttt{conf}(s_i > s_j)}{\sum_{s_t \in S_\sigma} \texttt{conf}(s_i > s_t)} \tag{8}$$

where S_σ is a song set and each s_t in S_σ forms a song difficulty ordering $s_i > s_t$.

4.2.3 Reliability

It is more reliable to make recommendation based on the song difficulty orderings which have both sufficient support and high confidence. We define the reliability of song difficulty ordering as follows.

Definition 5 (Reliability of Difficulty Ordering). *The reliability of the difficulty ordering $s_i > s_j$, denotes as $\texttt{Reli}(s_i > s_j)$, is the probability of the event that $s_i > s_j$ gets sufficient supports and a high confidence.*

We define the event that $s_i > s_j$ gets sufficient supports as $S(s_i > s_j)$, the event $s_i > s_j$ gets a high confidence as $C(s_i > s_j)$. As the definition, $\texttt{Reli}(s_i > s_j)$ equals to $P(S(s_i > s_j), C(s_i > s_j))$. We can suppose the two events are independent, then

$$P(S(s_i > s_j), C(s_i > s_j)) = P_{supp}(S(s_i > s_j)) * P_{conf}(C(s_i > s_j)) \tag{9}$$

Therefore,

$$\texttt{Reli}(s_i > s_j) = \frac{\texttt{supp}(s_i > s_j)}{\sum_{s_t \in S_\sigma} \texttt{supp}(s_i > s_t)} * \frac{\texttt{conf}(s_i > s_j)}{\sum_{s_t \in S_\sigma} \texttt{conf}(s_i > s_t)} \tag{10}$$

where S_σ is a song set and each s_t in S_σ forms a song difficulty orderings $s_i > s_t$.

The reliability of difficulty ordering will be used in the process of song recommendation. See Section 5 for details.

4.3 Discover Song Difficulty Orderings

One essential problem of discovering difficulty orderings is to obtain the difficulty voting introduced in Definition 1. Specifically, we need to obtain the number n_{ij} that support $s_i > s_j$ and n_{ji} that opposes the ordering. To this end, we exploit the users' ratings which implicitly reflect singers' opinions on song difficulty.

Each user of the social singing community is considered a singer. We first deduce the singer's voting for song pair s_i and s_j as follows. Suppose user u_k has sang s_i and s_j, if the listeners' mean ratings for the two songs is $r_i^k < r_j^k$. We know the listeners will give a high rating to the song if the user performs well. So the rating means u_k perform s_j better than s_i. This reflects s_i is more difficult than s_j for u_k. We can deduce u_k's voting preference for songs s_i and s_j is $s_i > s_j$.

We discover song difficulty orderings from user's singing history as follows.

1. Discover the song pairs that are sung by at least θ users.
2. For each song pair s_i,s_j, we deduce each singer's voting from the rating and count the number of votes $s_i > s_j$ and $s_j > s_i$ get as n_{ij} and n_{ji} respectively.
3. Set the one in $s_i > s_j$ and $s_j > s_i$ with more votes as s_i,s_j's song difficulty ordering.

Knowing the difficulty voting results of difficulty orderings, we can also calculate the reliability of the discovered song difficulty orderings.

5. SONG RECOMMENDATION BASED ON DIFFICULTY ORDERINGS

In this section, we propose a song recommendation approach based on the difficulty ordering introduced in the previous section. The basic idea of our approach is to estimate the likelihood that a user can perform the songs well, called *performance degree*. The estimated performance degree is then used for song recommendation. For instance, if we know that the user has larger performance degree on song s_i than s_j, then we would prefer s_i for the recommendation. In this approach, to recommend songs to user u_k, we first estimate user's performance degree on each song outside her singing history (i.e., $S - S_k$), and then rank the songs based on the performance degree as the final recommendation list.

As mentioned in Section 4, we estimate the performance degree based on the song difficulty orderings. Specifically, we first pick the songs in user u_k's singing list S_k that are evaluated by listeners as having been well-performed as *seeds*. Then, we exploit the difficulty orderings to find the songs which have large probabilities to be easier than the seeds, and utilize these probabilities to estimate the performance degree.

We formalize the above-mentioned idea in this section. We formally define performance degree and how we infer the performance degree in Section 5.1, and then develop an iterative algorithm in Section 5.2.

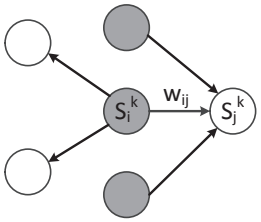


Figure 3: Inference on Difficulty Graph

5.1 Estimating Performance Degree

We formally define the performance degree, denoted by p_j^k, of user u_k on song s_j as the probability that u_k can perform well on s_j, i.e.,

$$p_j^k = Pr\{u_k \texttt{ performs well on } s_j\}$$

To estimate songs' performance degrees based on difficulty orderings. We first organize all difficulty orderings into an ordinary direct graph called *difficulty graph*. We denote the weight of edge from s_i to s_j in the difficulty graph as w_{ij}. We define difficulty graph as follow:

Definition 6 (Difficulty Graph). *A difficulty graph G is an ordinary directed weighted graph where the vertex[5] set S are songs, each edge from song s_i to s_j represents a song difficulty ordering with w_{ij} as the weight.*

Next, we present how to estimate performance degree p_j^k for song s_j from the performance degrees of its nearby songs through probabilistic inference. Specifically, suppose that the performance degrees of the songs in s_j's in-degree song set S_{in}^j are all known, shown in the gray vertices in Figure 3. We denote the event that u_k performs well on s_j as $W(s_j^k)$. Now we discusses how to infer p_j^k from the performance degree of the songs in S_{in}^j.

First we use sum and product rules of probability to get

$$Pr(W(s_j^k)) = \sum_{s_i^k \in S_{in}^j} [Pr(W(s_j^k)|W(s_i^k)) * Pr(W(s_i^k))] \quad (11)$$

We let

$$w_{ij} = Pr(W(s_j^k)|W(s_i^k)) \quad (12)$$

which is the edge weight from s_i to s_j. It represents the probability the user can perform s_j well, given that user performs s_i well. It equals to the probability that judgment $s_i^k > s_j^k$ is correct, i.e., the reliability of the song difficulty ordering $\texttt{Reli}(s_i^k > s_j^k)$. Then, we have

$$\begin{aligned} w_{ij} =& \texttt{Reli}(s_i^k > s_j^k) \\ =& \frac{\texttt{supp}(s_i^k > s_j^k)}{\sum_{s_t^k \in S_{out}^i} \texttt{supp}(s_i^k > s_t^k)} * \frac{\texttt{conf}(s_i^k > s_j^k)}{\sum_{s_t^k \in S_{out}^i} \texttt{conf}(s_i^k > s_t^k)} \end{aligned} \quad (13)$$

where S_{out}^i represents the out-degree vertex set of s_i, shown as the white vertices in Figure 3.

Knowing the performance degree of s_j's in-degree vertices and the edge weight, we can infer s_j's performance degree p_j^k. However, user only sings very few songs in the difficulty graph, how to infer s_j's performance degree if the performance degree of its in-degree vertices are unknown. In the next part, we will introduce an iterative algorithm to deal with this problem.

[5]We use vertex and song interchangeably

5.2 Iterative Probabilistic Inference

Based on the estimation method introduced in Section 5.1, we introduce a graph-based inference algorithm called *Iterative Probabilistic Inference (IPI)* to compute unknown performance degree of songs from the known performance degree of songs in the singing history.

We denote the performance degree of each vertex as $\mathbf{d} = [d_1, d_2, ..., d_{|S|}]$, where $|S|$ is the number of vertices in the difficulty graph. We construct the weight matrix $\mathbf{W}$ which is a $|S| \times |S|$ matrix, where W_{ij} equals to the weight from song s_i to s_j, that is $W_{ij} = w_{ij}$. Based on the performance degree estimation described in Section 5.1, we can perform probabilistic inference in the difficulty graph as follows

$$\mathbf{d}(t+1) = \mathbf{W}^T \mathbf{d}(t) \tag{14}$$

Because the initial performance degree $\mathbf{d}$ of each vertex is the same, each user's recommendation result will be the same. We need to give some "advantage" to user's previous singing songs. We construct a bonus vector as $\mathbf{b} = [b_1, b_2, ..., b_{|S|}]$ and assign bonus 1 to the songs that the user has sung in the past, $b_i = 1$, otherwise $b_i = 0$. The bonus vector allows user's previous singing songs to have a probability of getting an additional performance degree in each iteration. Through the iteration, these vertices will also affect other vertices' performance degrees.

Then the probabilistic inference in each iteration can be represented as

$$\mathbf{d}(t+1) = \alpha \mathbf{W}^T \mathbf{d}(t) + (1-\alpha)\mathbf{b} \tag{15}$$

where α represents the probability of inferring from the in-degree vertices, $1-\alpha$ as the probability of getting the bonus.

Then we repeatedly perform the probabilistic inference in the difficulty graph until the vertices' performance degrees converge. Theoretically, the iteration will converge to $(\mathbf{I} - \alpha * \mathbf{W}^T)^{-1} * \mathbf{b}$ according to [18]. Empirically, we observed that the iteration algorithm took three minutes to converge in our experiments. Because the calculation of edge weight in difficulty graph guarantees the sum of s_i's out-degree edge weight $\sum_{s_t \in S^i_{out}} w_{it} \leq 1$, we set $W_{ii} = 1 - \sum_{s_t \in S^i_{out}} w_{it}$ to make sure the iteration on the difficulty graph will converge. The complete algorithm description is given in Algorithm 1

Algorithm 1: *Iterative Probabilistic Inference*

```
Input: weight matrix W;bonus vector b
Output: final performance degree vector d
δ = 0.0001; error = ∞; t = -1;α ∈ [0, 1);
d where d_i = 1/sizeof(d)
while error > δ do
    t = t + 1;
    d(t+1) = αW^T d(t)+(1-α)b;
    error = d(t+1) - d(t);
d* = d(t)
```

6. EXPERIMENTS

In this section, we report the experimental setup and results. We first introduce the datasets being used in the experiments. After that, we report the baseline methods used to compare with our algorithm. Then we introduce the evaluation metrics used for measuring the performance of the algorithms. Finally, we present and analyse our experimental results.

6.1 Datasets

6.1.1 Song Difficulty Ordering Dataset

Since there is no publicly available dataset for singing-song recommendation, we have collected the data from 5sing which is the largest social singing community in China. In 5sing, users can upload their song recordings, listen and rate other user's recording. We chose this website because it has sufficient performance ratings for most users' singing songs.

For song difficulty orderings discovery, we crawl song pairs which have been sung by at least 10 users. The rating score for these song pairs must satisfy two conditions: the songs' rating scores in each song pair are non-zero and different. We model these song pairs into song difficulty orderings. Note that we will ignore song pairs whose ratings for both songs are the same. Finally we get a song difficulty ordering dataset. It contains total 5877 songs. We get totally 257666 song difficulty orderings.

6.1.2 Singing-song Recommendation Dataset

Now we describe the acquisition of our singing-song recommendation dataset. We download user's singing history from 5sing. Only songs appearing in the song difficulty orderings are kept. Because most of the users sing only a limited number of songs, we crawl the users who have sung at least 60 songs to partly overcome the data sparsity problem. Finally, we obtain a total of 2705 users. The number of songs sung by each user is 96. Despite this selection, the recommendation dataset is still very sparse. Note that the evaluation result of the recommendation for sparse dataset is usually not so high. For example, the best MAP reported by [12] is 0.048 which has 500 songs listened by each user in a 30520 song set. So we focus on comparing the relative increase in performance. We expect that in the future, the social singing community will develop the singing-song recommendation system and provides abundant data for the singing-song recommendation research.

For evaluation purposes, we randomly select 30% songs as the testing set, while the remaining one are used as the training set. We make sure there is no overlapping song between each user's training set and testing set.

6.2 Evaluation Metric

For evaluation, we use precision@n, recall@n and precision recall curve to compare the recommendation accuracy of our method with the baseline methods. We use mean average precision (MAP), Normalized Discount Cumulative Gain@n (NDCG@n) to measure ranking performance.

Precision@n and recall@n shows the precision and recall value at the specific position in the ranking list. Precision@n is defined as the number of correctly recommended songs divided by the total number of songs recommended at position n. While recall@n is defined as the correctly recommended songs divided by the total number of songs should be recommend in the test set.

We also use 11-point interpolated average precision to demonstrate the precision change at all recall levels. We plot the precision at 11 recall levels (0, 0.1, 0.2,..., 1.0) for the average of all users. For each user, the precision at each recall level i is determined by taking the maximum precision at any actual recall levels which are bigger than i. This is the precision recall curve for one user. After calculating the recall levels for each user, we average the precision at the corresponding recall levels over all users to get the final precision recall curve.

We use mean average precision (MAP) [2] to compare the ranking accuracy measured from the whole rank list. For each user, it first calculates the precision at the position of each relevant song in the recommend list. Then these precision values are reduced by their rank positions so that top ranked relevant documents contribute more to the final score. The sum of the precision values is averaged by the total number of correctly recommended items. Now we get the average precision (AP). MAP is calculated by averaging the AP value of each query.

$$MAP = \frac{1}{|Q|}\sum_{q\in Q}\frac{\sum_{n=1}^{N}(Precision@n * rel(n))}{R_q} \quad (16)$$

where Q is the test query set, R_q is q's relevant document, $rel(n)$ is a binary function on the relevance of a given rank, N is the number of retrieved documents.

We use NGCG@n [11] to measure the accuracy of the rank order at specific position of the ranking result. It calculates as follow

$$NDCG(n) = \frac{1}{Z_n}\sum_{j=1}^{n}\frac{2^{r(j)}-1}{log_2(j+1)} \quad (17)$$

where j is the predicted rank position , $r(j)$ is the rating value of j-th document in the ground-truth rank list, Z_n is the normalization factor which is the discounted cumulative gain in the $n-th$ position of the ground-truth rank list. We consider $r(j)$ as 1 if the song is sang by the user, otherwise 0.

6.3 Baseline Methods

We denote our iterative probabilistic inference algorithm as *IPI*. We compare IPI with four baselines.

The first classic baseline is the user-based *Collaborative Filtering (CF)* method. We build the user-song matrix using user's singing history. Songs sang by each user will have a 5-graded rating. Our objective is to predict the rating for other unrated songs. CF utilizes users' current song ratings and the similarity between users to predict the other songs' rating. We recommend songs which have high predicted rating scores. Suppose we want to get the predicted rating value for user u_t and song s_i, we get the rating value r_i^t as

$$r_i^t = \sum_{u_j\in S(u_t,k)\cap N(s_i)}\frac{Sim(u_t,u_j)}{\sum_{u_j\in S(u_t,k)\cap N(s_i)}Sim(u_t,u_j)} * r_i^j \quad (18)$$

Where $S(u_t,k)$ is the top k similar users with the target user u_t, $N(s_i)$ is the user set that has rating for song s_i, r_i^j is user u_j's rating for song s_i and $Sim(u_t,u_j)$ is the similarity between user u_t and u_j. We set the number of the nearest neighbor k to 20 to achieve the best performance. For the user's similarity, we use the cosine distance to calculate the similarity between two users' song rating vectors.

Another popular baseline is graph based song recommendation algorithm. We use the similar baseline as [7]. We build an undirected graph with song and user as vertex, user's singing relation as edge. We use the graph ranking algorithm proposed in [9]. We call this baseline as *Ordinary Graph based Recommendation (OGR)*.

Because measuring the reliability of song difficulty orderings is the key for singing-song recommendation. We compare the result of singing-song recommendation using only support or confidence to estimate the reliability of difficulty orderings. We denote these two baselines as *IPI-supp* and *IPI-conf* respectively.

For IPI-supp, the edge weight of difficulty graph can be calculated as

$$w_{ij} = \frac{\texttt{supp}(s_i > s_j)}{\sum_{s_t\in S_{out}^i}\texttt{supp}(s_i > s_t)} \quad (19)$$

For IPI-conf, the edge weight of difficulty graph can be calculated as

$$w_{ij} = \frac{\texttt{conf}(s_i > s_j)}{\sum_{s_t\in S_{out}^i}\texttt{conf}(s_i > s_t)} \quad (20)$$

6.4 Experimental Results

6.4.1 Baseline Comparison

We use the metrics mentioned in Section 6.2 to compare the performance of different algorithms. We fix the common random walk parameter α shared by OGR, IPI-supp, IPI-conf, IPI as 0.85. We estimate human's voting accuracy using the equation 7, we get $p = 0.71$. Figure 4 demonstrates the song ranking performance of the five algorithms.

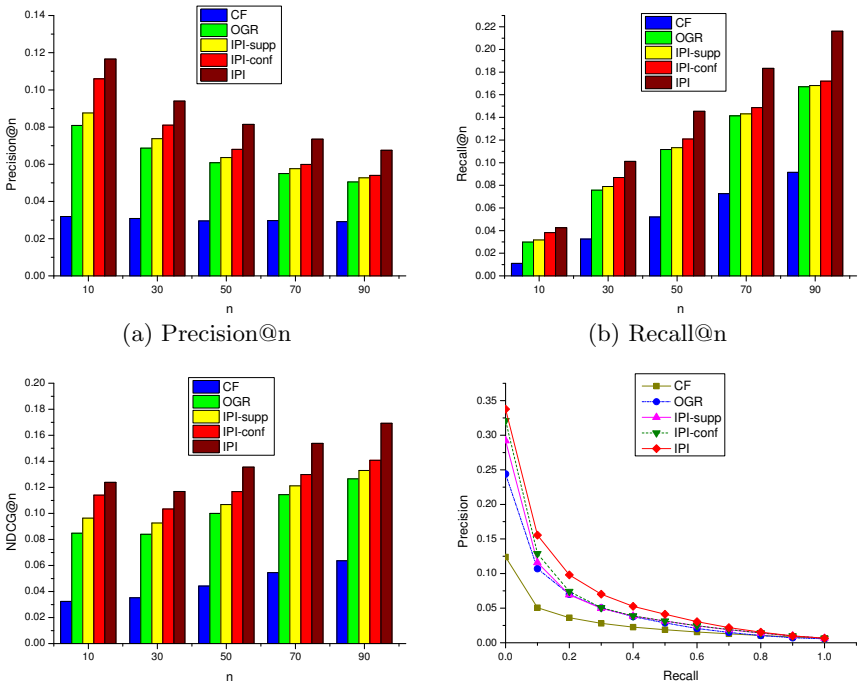
Figure 4(a) and Figure 4(b) are the precision and recall results of the five algorithms. We find our method IPI outperform baseline CF, OGR, IPI-supp and IPI-conf by an average of 180%, 36%, 28% 18% in precision and 190%, 32%, 29% 19% in recall. For IPI, IPI-supp, IPI-conf, those methods that make recommendation on difficulty graph perform better than the traditional song recommendation algorithms OGR and CF. This is because CF and OGR ignore the song's difficulty in performance during recommendation. It proves the effectiveness of song difficulty orderings in the singing-song recommendation. Figure 4(c) and Figure 4(d) are the performance results measured by NDCG@n and precision recall curve, it shows the similar results. IPI is always the best and those baselines who do recommendation on difficulty graph are also much better than the traditional song recommendation algorithms.

The above results show that IPI is better than IPI-supp and IPI-conf in all metrics. The difference between IPI and IPI-supp, IPI-conf is that the later only uses single factor support or confidence to estimate the reliability of the song difficulty orderings. This proves that using both factors to measure the reliability is better than the single one. This is reasonable because difficulty orderings with high support only represent the absolute number of people who support the difficulty orderings, it is possible that there are also many people who do not agree with the difficulty orderings. However, confidence considers the agreement of two sides, while ignoring the possible bias caused by the shortage of judgments. Adding both of them in reliability estimation overcomes the shortage of both factors.

Table 1 is the performance result measured from MAP. We find IPI outperform CF and OGR by an average of 160% and 40% respectively. But the absolute values are quite low, this is caused by the sparseness problem of the dataset. We compare the sparseness of our data set with the one reported in [12]. In their dataset, the number of songs listened by each user is 1.6% of the total song set. Their precision recall curve also shows a fast decrease in the precision and the best MAP is 0.048. [17] is another example. In our dataset, the number of songs sang by each user is also about 1.6% of the total song set. The low performance value is reasonable.

Table 1: Ranking accuracy in terms of MAP

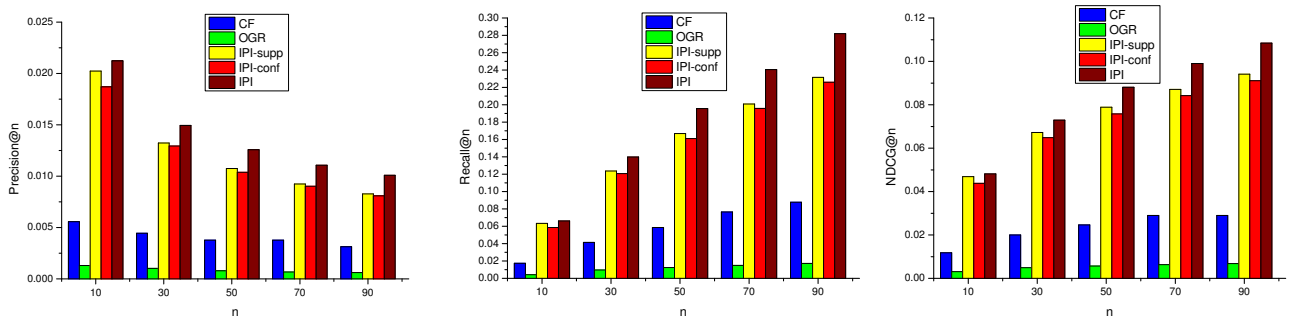
Algorithm	IPI	IPI-conf	IPI-supp	OGR	CF
MAP value	0.061	0.051	0.048	0.043	0.024



(a) Precision@n (b) Recall@n

(c) NDCG@n (d) Precision-recall Curve

Figure 4: Performance Comparison of the Five Algorithms



(a) Precision@n (b) Recall@n (c) NDCG@n

Figure 5: Performance Comparison for Cold Start User

6.4.2 *Test for Cold Start User*

We also test the performance of the algorithms for cold start users. We consider those users who sing between 7 to 10 songs in the training set as cold start users. Limiting the number of each user's singing-song is to ensure there will be at least 3 songs in the testing set for the evaluation purpose. The number of users in the sparse dataset is 3000.

Figure 5(a) and Figure 5(b) is the precision and recall results of the five methods on the sparse dataset. We find the performance of IPI-conf is lower than IPI-supp which is opposite to the above results. This is caused by the error in reliability estimation using only support factors. For those conjectures with few supports, even if the confidence is high, it still exhibits errors when estimating the reliability. For cold start users, they do not have enough previous singing songs as query to overcome the error cause by confidence factor. While support can also reflect the popularity of the song difficulty ordering. Large number of support means it is more possible that many people had sung the song pair. For cold start user, recommending popular songs through popular difficulty orderings may perform better. Our IPI again performs better than the two baselines which shows again the measuring the reliability through both the support and confidence is effective. Figure 5(c) is the performance measured from NDCG@n which shows the same results.

Table 2: Statistical significance comparison between IPI and two IPI variants

Precision@n	P@10	P@30	P@50	P@70	P@90
IPI-supp	1.384	5.2259	8.4185	11.0886	13.2275
IPI-conf	3.9934	6.5927	10.3389	12.7425	15.097

Because the precision differences between IPI and two IPI variants are minor in the cold start dataset. We conduct the significance test(t-Test) on the results. For each method, we compute a list of precisions@n, each corresponding to the recommendation for a user. Given IPI and a baseline(IPI-supp or IPI-conf), we compute t-ratio over their precision lists. Table 2 shows the t-ratios between IPI and two baselines. We find most t-ratios are larger than the cutoff of 3.3 (nearest df in t-test table is 1000 under $p<0.001$), which shows IPI's improvement is significant. The only exception

is the result of the precision@10 for IPI-supp, which is caused by the data sparse problem.

For the baseline OGR, the performance drops a lot on the sparse dataset. It is caused by the existence of large amounts of isolated vertices in song-user graph. Due to the sparseness of the dataset, many songs in the test set had not been sung by any user in the train set. It is impossible to be recommended based on the song-user graph. By looking into the graph, we find 1391 out of 5877 songs had not been sung by any user in the training set. While our difficulty graph will not be affected by this data sparseness problem. This is because our difficulty graph is pre-built, the iterative algorithm will not largely be affected by the shortage of user's singing relations.

7. CONCLUSIONS

In this paper, we address the singing-song recommendation problem in social singing communities. We propose a method to discover song difficulty orderings from the song performance ratings of each user. We also estimate the reliability of the discovered difficulty orderings based on the support and confidence of the difficulty orderings. For singing-song recommendation, we model the discovered difficulty orderings as a difficulty graph that captures the difficulty relation in performance between different songs. An iterative probabilistic inference algorithm is proposed to estimate the performance degrees of all the other vertices in the graph from each user's past singing history. The experiments have been performed on a dataset collected from an actual social singing community (5sing). The result shows our proposed framework is more effective than the traditional song recommendation methods.

For future work, we will conduct user study to see if the amateur singers are satisfied with the recommendation generated by this system. Secondly, because the discovered song difficulty orderings can also be used in other social singing communities, we will focus on discovering more difficulty orderings from other sources of social singing community to maintain a complete difficulty orderings database. Another direction for singing song recommendation is that we will exploit other factors such as users' singing preference as well as friendship relations to see if they can help make the recommendation better.

8. ACKNOWLEDGEMENT

This research was carried out at the NUS-ZJU SeSaMe Centre. It is supported by the Singapore NRF under its IRC@SG Funding Initiative and administered by the IDMPO. This research was also supported by the National High Technology Research and Development Program of China (Grant No. 2014AA015205).

9. REFERENCES

[1] S. Agarwal. Ranking on graph data. In W. W. Cohen and A. Moore, editors, *ICML*, volume 148, pages 25–32. ACM, 2006.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[3] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for youtube: Taking random walks through the view graph. WWW '08, pages 895–904, New York, NY, USA, 2008. ACM.

[4] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. volume 3120, pages 624–638. Springer, 2004.

[5] M. Belkin and P. Niyogi. Semi-supervised learning on riemannian manifolds. *Machine Learning*, 56(1-3):209–239, 2004.

[6] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *CoRR*, abs/1301.7363, 2013.

[7] J. Bu, S. Tan, C. Chen, C. Wang, H. Wu, L. Zhang, and X. He. Music recommendation by unified hypergraph: Combining social media information and music content. MM '10, pages 391–400, New York, NY, USA, 2010. ACM.

[8] D. Goldberg, D. A. Nichols, B. M. Oki, and D. B. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[9] T. H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.

[10] K. Hoashi, K. Matsumoto, and N. Inoue. Personalization of user profiles for content-based music retrieval based on relevance feedback. In *ACM Multimedia*, pages 110–119, 2003.

[11] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48, 2000.

[12] I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation. SIGIR '09, pages 195–202, New York, NY, USA, 2009. ACM.

[13] K. Mao, X. Luo, K. Chen, G. Chen, and L. Shou. myDJ: recommending karaoke songs from one's own voice. In *SIGIR*, page 1009. ACM, 2012.

[14] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295. ACM, 2001.

[15] L. Shou, K. Mao, X. Luo, K. Chen, G. Chen, and T. Hu. Competence-based song recommendation. In *SIGIR*, pages 423–432. ACM, 2013.

[16] X. Wu, Q. Liu, E. Chen, L. He, J. Lv, C. Cao, and G. Hu. Personalized next-song recommendation in online karaokes. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 137–140, New York, NY, USA, 2013. ACM.

[17] M. Ye, P. Yin, W.-C. Lee, and D. L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *SIGIR*, pages 325–334. ACM, 2011.

[18] D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *ICML*, volume 119, pages 1036–1043. ACM, 2005.

[19] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.