Music Playlist Generation based on Community Detection and Personalized PageRank

Final Report

CS 224W: Social and Information Network Analysis
Autumn 2015

Bangzheng He
Institute for Computational and Mathematical Engineering
Stanford University
bzhe@stanford.edu

Yandi Li
Department of Electrical Engineering
Stanford University
yandi@stanford.edu

Bobby Nguy
Department of Statistics
Stanford University
cnguy002@stanford.edu

Github: https://github.com/dhanian/playlist

1 Introduction

In the age of internet radio and digital music storage, millions of songs are available to listeners at a click of a button. With this huge amount of songs to choose from and with the taste of music varying from person to person, it becomes a difficult task for the user to select the "right" song for the right moment. Various playlist generation algorithms have been developed in which playlists are lists of sequentially ordered tracks, represent a possible solution to this issue and help users explore this huge collection of songs available. However the current schemes mostly focus on finding the closest songs to a seed song. We feel that a better playlist should not only play the closest songs but also serve to make it a discovery experience with lots of varieties. Therefore we will first study the community structure of our song similarity network. Then we will apply personalized PageRank with consideration of the community structure to generate playlist.

2 Review of prior work

There are different methodologies when it comes to music playlist generation. We summarize some of the approach as following:

Local search based approach [1]. This method aims to search the neighboring songs to optimize a cost function given by the total weighted penalty of the playlist.

Constraint based approach [2]. This is based on the local search approach but adds user-specific constraints like song categories and user preferences, which makes it a constraint satisfaction programming.

Content based approach [3]. In this approach, each song is represented as a vector of attributes like artist, composer, genre, mood etc. The constraints provided by the user or derived by the system will depend on the values of the attributes.

Similarity based approach [4]. An anonymous list of similar songs is generated according to the user input query. This is a suggestion of what other songs are similar to the one user provide.

Our interest is in the similarity based approach which is also most popular among the current music radio services. The conventional similarity based approach usually involves finding songs in a sequence that are most similar to the query songs. This can be mapped to a traveling salesman problem finding a playlist such that consecutive pieces are maximally similar. However, we think the recommendation works just like the web search. Therefore we implemented an unconventional Personalized PageRank approach to generate playlist. We also want to incorporate ideas from the constraint based approach and content based approach, therefore we detect communities from the music similarities network and improve our playlist with more variety. To do this, we borrow the idea of the Topic-Sensitive PageRank [5]. In Topic-Sensitive PageRank, a set of PageRank vectors is computed offline, biased using a set of representative topics. At query time, by using linear combinations of these precomputed biased PageRank vectors, a context-specific importance scores for pages can be generated more accurately and efficiently. Later we will show that, in our network the topics (communities) are too many to be computed offline. So instead, we will modify the personalization vector at query time according to the community structure.

3 Data collection and Preprocessing

There are many music services that gather their data about music in very different ways. Below is a table of some of the popular services and their possible source of data.

Service	**Source of data**
Pandora	Musicologists take surveys
Songza	Editors or music fans make playlists
Last.fm	Activity data, tags on artists and songs, acoustic analysis
All music guide	Music editors and writers
Amazon	Purchase and browsing history
iTunes Genius	Purchase data, activity data from iTunes
Echo Nest	Acoustic analysis, text analysis

In our project we choose to use Last.fm data prepared by Million Song Dataset project [6]. The Million Song Dataset team partnered with Last.fm to build this large collection of song-level tags and precomputed song-level similarity for research purpose. It has

- 943,347 tracks
- 584,897 tracks with at least one similar track
- 663,234 tracks with at least one in or out degree
- 56,506,688 (track - similar track) pairs

The precomputed song similarity from Last.fm is calculated through collaborative filtering and acoustic analysis. We will not argue about the "correctness" of the similarity measurement here but instead focusing on generating better playlist from the music similarity network. It is worth noting that according to their similarity measure the edge between two tracks might have different value from each direction. Our guess is that the similarity score is local to the chosen tracks, not normalized across the whole dataset. Hence the music similarity network will be a directed network, which actually makes our approach more suited than conventional similarity based approaches.

The data are provided to us in two formats. A raw data consist of one JSON file for each track. Keys are **artist**, **title**, **timestamp**, **similars** and **tags**. And a SQLite database *lastfm_similars.db* with two tables similars_src and similars_dest.

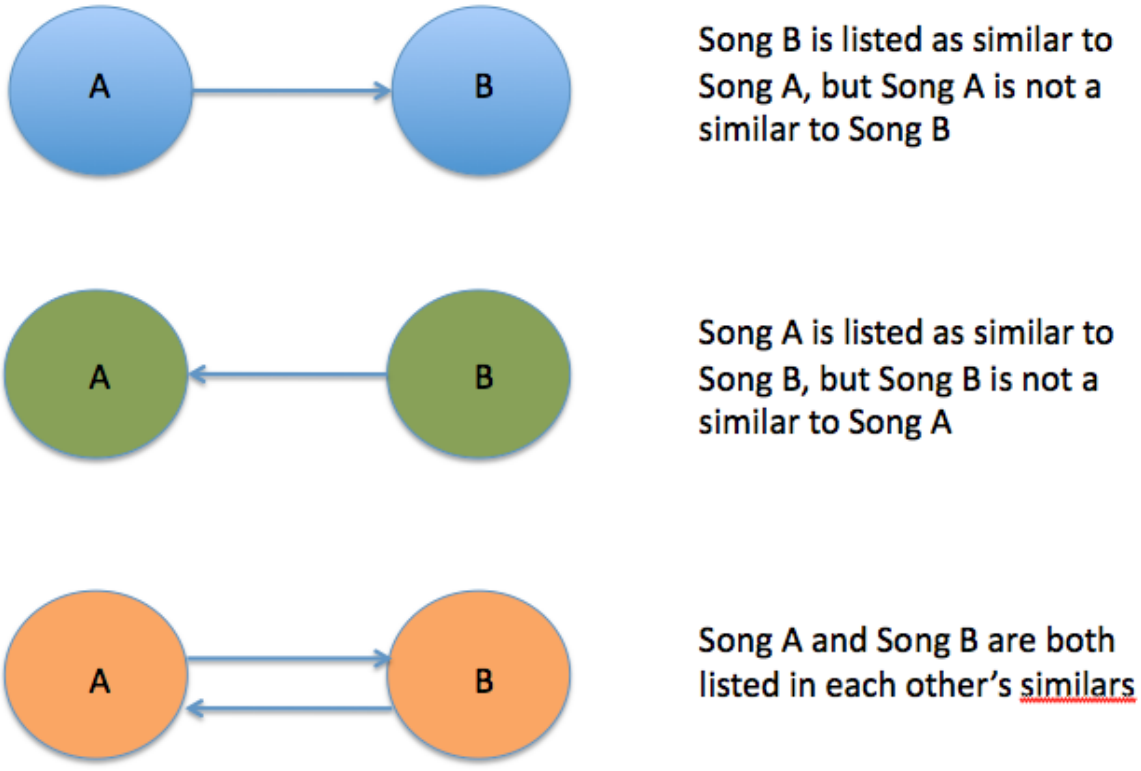
Since going over JSON files is inefficient, we choose to use the SQLite databases. The SQLite database is about 4.1GB large in storage, and records information of track ids and their similar songs' similar scores. The similars_src table has headers *tid* and *target* where *tid* are the unique track ids for each songs with at least one similar track and *target* are the track ids and similarity scores of its similar tracks. The similars_dest table has similar structure but the *target* are tracks that consider the chosen track as similar. We need only one of these table to build our music similarity network. In our case, we choose to use the similars_src table. We utilize SQLite's python interface to manipulate the dataset.

One problem of the database is that we do not have a full list all tracks. The similars_src table has 584,897 rows which represents all songs with at least one similar track. There are other songs which do not have any similar tracks but have other tracks consider them as similars. Since our dataset is very large and we need to be extra careful about memory usage, we choose to preprocess

the data to find the track ids of all 663,234 tracks with at least one in or out degree and index those track ids from 0 to 663233 and store them in a new SQLite database *songs_index.db*. For the other $943347 - 663234$ tracks, they are presented as JSON files but not in the SQLite database *lastfm_similars.db*. Since they are all isolated tracks with no edges connected to any other track, we choose to not use them at all.

4 Overview of the underlying network

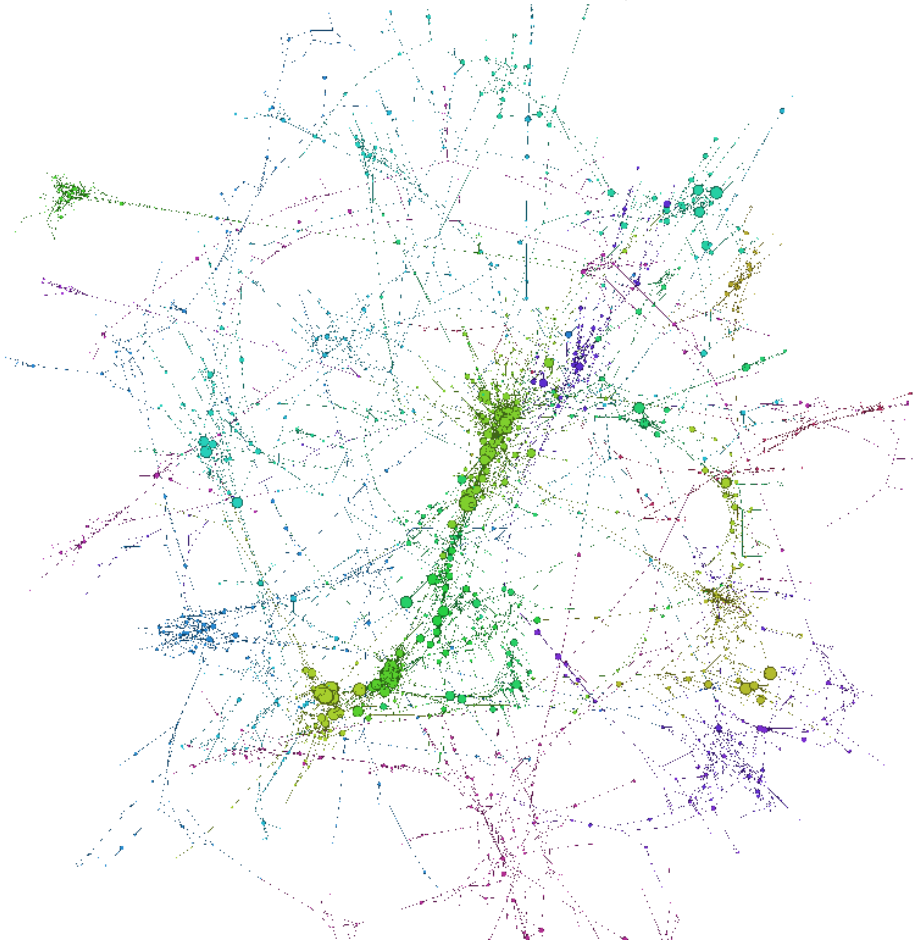
In our analysis, the song similarity network is casted as a directed graph. The directedness of the graph is a result of last.fm's methodology in calculating the similarity scores. These scores are computed based on two parts: acoustic analysis and users' listening data. For example, users who listen to song A often listen to song B, but users who listen to song B never listens to song A. Coupled with the level of acoustic (dis)similarity, song A would point to song B but not the other way round. To summarize, we would expect the following relationship between nodes in our graph:



We plotted a subset of the network graph with Gephi, with node size indicating the in-degree of the song. From this We can see a small set of songs capturing a lot of degrees, while the majority of other nodes stay relatively small in node size. To summarize with numbers, the degree centrality ranges from 1-300+, but the average degree in the graph is only around 10. There seems to exist a preferential effect on a small set of "popular" songs.

Another observation is that many of the node linkages seem to form a long chain. This is rather interesting because it suggests that majority of song writings are influenced by a rather localized selections of songs (overlooking the small set of high-degree nodes). The similarity/influ-

ence, however, can be propagated along a chain of many songs. This does have an implication in song recommendation. For example, it might be better to recommend songs of all the immediate neighbors of the current node first, instead moving in a single direction. This chain structure also makes the graph looks somehow decentralized. In the next section, however, we will see that we are able to get satisfactory results from some community detection algorithms. (The graph below is partitioned and colored by Modularity using Louvain algorithm.)



5 Community Detection

For community detection, there are three common used algorithms, Girvan-Newman community detection algorithm, Clauset-Newman-Moore algorithm and the Louvain method. And the last two are widely used for very large networks.

The GirvanNewman algorithm [7] detects communities by progressively removing edges from the original network. The connected components of the remaining network are the communities. Instead of trying to construct a measure that tells us which edges are the most central to communities, the GirvanNewman algorithm focuses on edges that are most likely "between" communities, meaning this algorithm optimizes the community structure according to betweenness.

Clauset-Newman-Moore algorithm [8] is a hierarchical agglomeration algorithm for detecting community structure which is very efficient: its running time on a network with n vertices and m

edges is O(md log n) where d is the depth of the dendrogram describing the community structure. Many real-world networks are sparse and hierarchical, with m n and d log n, in which case our algorithm runs in essentially linear time, O(n log2 n). The core of the algorithm is to greedily optimize modularity.

The Louvain method [9] is a simple, efficient and easy-to-implement method for identifying communities in large networks. The method is a greedy optimization method that attempts to optimize the modularity" of a partition of the network. The optimization is performed in two steps. First, the method looks for "small" communities by optimizing modularity locally. Second, it aggregates nodes belonging to the same community and builds a new network whose nodes are the communities. These steps are repeated iteratively until a maximum of modularity is attained and a hierarchy of communities is produced. Although the exact computational complexity of the method is not known, the method seems to run in time O(n log n) with most of the computational effort spent on the optimization at the first level. Exact modularity optimization is known to be NP-hard. By comparing these three methods, we can see that the GirvanNewman algorithm is not a wise choice for our project as our music network is a music graph with 1 million nodes and average degree is about 10. For the other two, the Louvain seems faster as its complexity is only O(n log n), slightly better than O(n log2 n).

Considering the size of our network (663,234 nodes, 56,506,688 edges), we choose Louvain method which is fastest and most memory efficient. Noting that for our original data, as there are too many similar songs with very low similarity score, to make the calculation plausible for a single laptop with 8GB RAM, we decided to only include edges with similarity scores greater than 0.5 for community detection. With this set-up, the Louvain method gave us a partition with 91,897 communities, and the modularity of this partition is 0.918.

6 Personalized PageRank

PageRank was initially introduced to rank web pages, but since our playlist generation resembles a web search, we choose to utilize personalized PageRank for sequencing our playlist.

In a standard weighted PageRank, we consider a random walk on a graph with weighted adjacency matrix $A \in R^{n \times n}$, in which a nonzero entry $A_{ij} > 0$ represents the edge weight from node i to node$j$. The weighted out-degree of node i is $d_i = \sum_j A_{ij}$, and we define the diagonal matrix D with diagonal entries d_i. If there's no sinking(nodes with zero outgoing weight), the transition matrix for a random walk on this weighted graph is $P = D^{-1}A$, which is a row-stochastic matrix. The random walker at each step moving to a new node by transitioning along the edge with probability α or by teleporting to a position independent of the previous location with probability $1 - \alpha$. Then we can write the Markov process as:

$$x^{\intercal(t+1)} = \alpha x^{\intercal t} P + (1 - \alpha) v^{\intercal}$$

where P represents the edge transition probabilities and the personalization vector v represents teleportation probabilities. The PageRank vector x is the stationary vector for the above process. When there's sinking, we give outedge to those sinking nodes pointing to the personalization vector.

When we choose different tracks to start the playlist we will have different personalization vector $v(w)$, we can weight the nodes according to the community structures of the network. Right now we solve the problem of calculating the PageRank vector x given a specific personalization vector v. We choose to use power iteration method utilizing scipy's sparse matrix implementation.

Our transition matrix is 663234×663234, which makes it impossible for computing the inverse matrix. Luckily, the transition matrix is very sparse, it has only 56506688 nonzero entries, which is about only 0.013% nonzero entries. Given our size of the dataset and sparse nature of our transition matrix P, it is reasonable to choose power iteration and use sparse matrix computation to speed up the program and save memory.

The power iteration is simply repeatedly calculating the equation above for new $x^(t+1)$ with any starting value of x^0 until the $l1$ norm of $(x^{(t+1)} - x^t)$ smaller than a tolerance value. For the sparse matrix computation we choose to store the sparse matrix in Compressed Row Storage(CSR) format and use scipy's sparse matrix multiplication. The maximum iteration is set to 100, but the iteration usually converges in 10 to 20 iterations.

The personal laptop computer we are using for our program has a $2.00GHz \times 4$ processor and an 8GB RAM. By pre-computing the CSR format matrix, in average we can generate a new playlist according to new user query in less than 20 seconds.

7 Playlist Generation

In order to combine our insights in community detection and personalized page rank to create a diversified and user-custom play list, we need to utilize the community structure of the network to modify the personalization vector at query time, then use Personalized PageRank to generate playlist.

To do this, we assume S is a set of songs that user inputs at query time. We will also have a parameter called "discovery rate" β ranging from 0 to 1, which controls how diversified the playlist will be. Assume the number of all songs(nodes) is $N = 663,234$. Denote the union of the communities that songs in S belongs as C_S. Then for the personalization vector:

$$v = (v_1, v_2, \ldots, v_N)$$

we set:

$$v_i = \frac{\lambda \times \beta}{N}, v_i \notin C_S$$

$$v_i = \frac{1}{N}, v_i \in C_S - S$$

$$v_i = \frac{|C_{v_i}|}{N}, v_i \in S$$

where λ is a scaling factor for the discovery rate. For our network with 663,234 nodes, we empirically tested and chose it to be 0.01. And $|C_{v_i}|$ is the size of the community node v_i is in. Then normalize v, use it as the personalization vector for the Personalized PageRank.

With this set-up, when the discovery rate β is set to 0, the chance to teleport to communities outside of the query songs are 0 in the Personalized PageRank, and the chance to teleport the query song is proportional to the size of the community it resides in to emphasize preference to the query songs over other songs in the community. As a result, the generated playlist is suit for user who just want to have a playlist with the most similar and "important" songs. In contrast, when the discovery rate β is set to 1, the chance to teleport to communities outside of the query songs is raised to have fairly large impact on the final results. As a result, the generated playlist is more diversified in regard to communities, which is suited for user who want to have a playlist that plays similar songs but also help to discover other music.

We can also incorporate user feedback into the playlist generation. Similarly, we modify the personalization vector according to user feedback. For example, if user skips a song, we set the corresponding v_i in the personalization vector to 0; if user likes a song, we set the corresponding v_i the same weight as if it was a query song.

8 Results and evaluation

There's no guideline for a good playlist or bad playlist, which makes it extremely hard to judge a playlist generation algorithm. Here we test it against our own liking of music and ask our friends for their opinions, parameters like scaling factor λ will also have large impact on the resulting playlist. Even with these uncertainties, the idea of generating playlist based on community and Personalized PageRank is promising and gives a new insight into playlist. To have a rough idea of how the playlist performs, here we provide an example of playlist generated with a well known song Britney Spears' "Lucky" choosing different discovery rate. And compare it with the playlist popular internet radio services generated. It is best to compare with the last.fm since we are using last.fm data, but unfortunately last.fm is not providing radio service anymore. Hence we compare our playlist with Pandora. We can see that the Pandora playlist consists of mainly popular songs

Our playlist			Pandora
Discovery Rate 0	Discovery Rate 0.5	Discovery Rate 1	
Britney Spears: Lucky	Britney Spears: Lucky	Britney Spears: Lucky	Christina Aguilera: Genie In A Bottle
Britney Spears: Stronger	Britney Spears: Stronger	Britney Spears: Stronger	*Nsync: It's Gonna Be Me
Britney Spears: Sometimes	Britney Spears: Sometimes	Alexandra Burke: Broken Heels	Backstreet Boys: Quit Playing Games
Britney Spears: Thinkin' About You	Britney Spears: Thinkin' About You	Cheryl Cole: Parachute	Britney Spears: (You Drive Me) Crazy
Britney Spears: The Beat Goes On	Cheryl Cole: Parachute	Oliver Nelson: Cascades	Christina Aguilera: What A Girl Wants
Christina Aguilera: Not Myself Tonight	Alexandra Burke: Broken Heels	Local Natives: Sun Hands	Spice Girls: Wannabe
Britney Spears: E-Mail My Heart	Christina Aguilera: Not Myself Tonight	Britney Spears: Sometimes	Backstreet Boys: I Want It That Way
Cheryl Cole: Parachute	Cheryl Cole: Fight For This Love	Cheryl Cole: Fight For This Love	Britney Spears: Everytime
Britney Spears: I Will Be There	Britney Spears: The Beat Goes On	Christina Aguilera: Not Myself Tonight	S Club 7: Never Had A Dream Come True
Britney Spears: I'm A Slave 4 U	Selena Gomez & The Scene: Naturally	Marco Polo & Torae: But Wait	*Nsync: I want You Back
Cheryl Cole: Fight For This Love	Lindsay Lohan: Rumors	Oliver Nelson: Yearnin'	Stacy's Mom: Fountains of Wayne
Britney Spears: Born To Make You Happy	Sugababes / Gracious K: About A Girl	Selena Gomez & The Scene: Naturally	Britney Spears: Crazy
Selena Gomez & The Scene: Naturally	Kristinia DeBarge: Goodbye	Oliver Nelson: Butch And Butch	Backstreet Boys: As Long As You Love Me
Alexandra Burke: Broken Heels	Leona Lewis: I Got You	Britney Spears: Thinkin' About You	Michelle Branch: Everywhere
Britney Spears: From The Bottom Of My B	Pixie Lott: Gravity	Marco Polo & Torae: Danger	Avril Lavigne: Sk8er Boi
Lindsay Lohan: Rumors	Girls Aloud: Call The Shots	Ra Ra Riot: Can You Tell	Britney Spears: Sometimes
Britney Spears: Do Somethin'	Oliver Nelson: Cascades	The Morning Benders: Promises	Backstreet Boys: Shape Of My Heart
Britney Spears: Oops!...I Did It Again	Britney Spears: I'm A Slave 4 U	Sugababes / Gracious K: About A Girl	Shaggy: Angel
Britney Spears: Everytime	Britney Spears: E-Mail My Heart	Texas In July: Lancaster	Christina Aguilera: Beautiful
Britney Spears: Soda Pop	Britney Spears: I Will Be There	Leona Lewis: I Got You	Vanessa Carlton: A Thousand Miles
Sugababes / Gracious K: About A Girl	Britney Spears: Born To Make You Happy	Big L: Put It On	Avril lavigne: Complicated
Britney Spears: ...Baby One More Time	Local Natives: Sun Hands	Surfer Blood: Floating Vibes	Celine Dion: My Heart Will Go On
Kristinia DeBarge: Goodbye	Esmée Denters: Outta Here	Lindsay Lohan: Rumors	Christina Aguilera: Fighter
Britney Spears: My Prerogative	Jordin Sparks: Battlefield	Two Door Cinema Club: This Is The Life	Destiny's Child: Survivor
Britney Spears: (You Drive Me) Crazy	Katy Perry: I Kissed A Girl	Joe Henderson: Isotope	*Nsync: Pop
Pixie Lott: Gravity	Britney Spears: From The Bottom Of My B	Kristinia DeBarge: Goodbye	Usher: U Got It Bad
Britney Spears duet with Don Phillip: I Wil	Alesha Dixon: The Boy Does Nothing	The Wombats: Moving To New York	Play: I Must Not Chase The Boys
Hilary Duff: Reach Out	Hilary Duff: Reach Out	Surfer Blood: Harmonix	Michael Jackson: Immortal
Katy Perry: I Kissed A Girl	Britney Spears: Do Somethin'	Pixie Lott: Gravity	Uncle Kracker: Follow Me
Girls Aloud: Call The Shots	Britney Spears: Everytime	Girls Aloud: Call The Shots	Glee Cast: We Are Young

around the same age as "Lucky". It is probably a result of a more complex model which focuses more on artist similarity and also incorporate other song contents like year of release into consideration. In comparison, our playlist is sololy based on song similarity. As simple as it is, it still shares some of the common songs with the Pandora playlist. And by changing discover rate, user can control the variety brought to their playlist.

9 Conclusion

Even though our generated playlist can't compete with the commercial internet radio playlist for now, the idea of utilizing song similarity through network analysis, for example community detection and Personalized PageRank, is promising and does provide new insight into playlist generation. In the future work, we can incorporate more information of the songs, like artist similarity, year, composer into this basic model and fine tune the function for personalization vector.

References

[1] Steffen Pauws, Wim Verhaegh and Mark Vossen, *Fast Generation of Optimal Music Playlists using Local Search*, Proceedings of the 6th International Conference on Music Information Retrieval(ISMIR), 2006

[2] Pachet F, Roy P and Cazaly D, *A combinatorial approach to content-based music selection*, Proceedings of IEEE multimedia computing and systems international conference, Firenze, pp 457462, 1999.

[3] Martin Gasser, Elias Pampalk, and Martin Tomitsch, *A Content-Based User-Feedback Driven Playlist Generator and its Evaluation in a Real-World Scenario*, The Australian Research for Artificial Intelligence, 2008.

[4] Crampes M, Villerd J, Emery A and Ranwez S, *Automatic playlist composition in a dynamic music landscape*, Proceedings of the international workshop on semantically aware document processing and indexing, 2007.

[5] T. H. Haveliwala. *Topic-Sensitive PageRank*, 11th International World Wide Web Conference, 2002.

[6] Last.fm dataset, the official song tags and song similarity collection for the Million Song Dataset, available at: http://labrosa.ee.columbia.edu/millionsong/lastfm

[7] Girvan M. and Newman M. E. J., *Community structure in social and biological networks*, Proc. Natl. Acad. Sci. USA 99, 78217826 (2002).

[8] Clauset, Aaron, Mark EJ Newman, and Cristopher Moore, *Finding community structure in very large networks*, Physical review E 70.6 (2004): 066111.

[9] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, *Fast unfolding of communities in large networks*, Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P10008 (12pp)

Bangzheng He: Problem formulation, algorithm and coding of Personalized PageRank and playlist generation, testing, report

Bobby Nguy: Plotting graphs during data analysis, crawling and preprocessing the data, preliminary data analysis and overview of underlying network, report, poster

Yandi Li: Problem formulation, algorithm and coding of community detection, report

---