Similarity Measures in the World of Music

Master Thesis

Michael Lorenzi
<mlorenzi@student.ethz.ch>

Advisors:
Prof. Dr. Roger Wattenhofer
Olga Goussevskaia
Michael Kuhn

Distributed Computing Group
Computer Engineering and Networks Laboratory (TIK)
Department of Computer Science

August 19, 2007

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group

Abstract

The distribution of music happens more and more over the internet. On single computers large collections of music are formed. And they are ever growing since the copying of digital data is an easy obstacle to overcome. These collections are so large that one hardly can have an overview of it. To help ordering music, in this thesis musical songs are encoded in a graph representation. Songs which are linked are expected to be similar. The distance in the graph then is the similarity of two songs. A large collection of user's favorite songs is downloaded from last.fm to form such a graph with 430000 nodes. Furthermore, to evaluate the quality of graph and embedding, a measurement using a category-tree is introduced. Most algorithms on the raw graph are too complex (even a single shortest-path-computation has complexity $O(m + n * \log n)$). To create a reasonable application using this graph, it is therefore almost inevitable to assign distance labels to the nodes. A new embedding-algorithm (IterativeLMDS) is proposed which is based on LMDS, and it is shown that it improves the quality. Using the embedding, a web-application is presented which is able to create playlist and propose styles for songs in less than a second.

Musik wird immer mehr digital über das Internet verbreitet. Auf einzelnen Computer sammeln sich grosse Mengen an Musikstücken an. Und diese Musik-Kollektionen wachsen ständig, da das Kopieren und Verteilen von digitalen Daten sehr einfach ist. Die Menge an Musikstücken wird so gross, dass es zunehmend schwieriger wird, den Ueberblick darüber zu behalten. In dieser Arbeit werden Musikstücke in Form eines Graphen dargestellt, um die Ordnung und die Organisation von Musik zu vereinfachen. Die Distanz in diesem Graphen entspricht dabei der Aehnlichkeit der beiden entsprechenden Musikstücke. Aus einer grosse Anzahl von Lieblingssongs von last.fm-Usern wird ein Graph mit etwa 430000 Knoten gebildet. Um die Qualität dieses Graphen und der Embeddings zu messen, wird ein Mass für die Qualität von Aehnlichkeit von Musik, basierend auf einem Kategorien-Baum, eingeführt. Da die meisten Algorithmen für Graphen zu komplex sind (die Berechnung eines kürzesten Pfades hat Komplexität $O(m+n*\log n)$)), ist es beinahe unabdingbar, ein Embedding des Graphen erzeugen. Dazu wird ein neuer Embedding-Algorithmus (IterativeLMDS) vorgeschlagen der auf LMDS basiert. Es wird gezeigt, dass die Qualität des Embeddings durch IterativeLMDS erhöht wird. Unter Verwendung des Embeddings wird eine Web-Applikation entwickelt, mit der man Playlisten generieren kann und Musikstücken Genres zuordnen kann. Dank dem Embedding dauern diese Abfragen nur wenige Zehntelsekunden.

Contents

1 Introduction

1.1 Task Description

The task of this master thesis can be divided into two parts:

- Create a music-graph with either songs or artists as nodes. Edges should represent some kind of relationship or similarity.
- Develop an interesting application using this graph

To create such a music-graph there are several challenges and problems to solve, such as finding an appropriate source of information about music similarity (see Chapter 3) or dealing with large amounts of data.

1.2 Motivation

The sales figures of conventional physical music stores are declining in the past few years. The distribution of music happens more and more over the internet. Music is downloaded directly to computers, and from there copied to mobile music-players or streamed to other devices. On single computers large collections of music are formed. And they are growing since the copying of digital data is an easy obstacle to overcome. These collections are so large that one hardly can have an overview of it. The organization of these large collections is an important and challenging topic in music distribution.

The organization in folders soon came to limits. Today, music is mostly organized with metadata. A notion of similarity of music would certainly help organizing and retrieving music. It could ease the handling of large music collections and give recommendations of what fits together.

1.3 Challenges

The world of music is expected to be huge. It goes back to the time of the great composers like Mozart or Beethoven and ranges to the stars like Elton John or Mariah Carey. Every day new songs are produced. It contains a lot of genres and styles like classical, rock n' roll or rap to mention just a few. Thus, computing a graph which contains the whole world of music is quite challenging, and the quality and the up-to-dateness of this graph can not be foreseen before making first tryouts.

Another challenging aspect of this task is the "human factor". Which music sounds similar, or which songs are completely different, depends significantly on the personal background of the listener. One person could weight the voice of the lead singer as primary attribute for similarity, another could compare the used musical instruments, and yet another compares social factors such as the message delivered in their lyrics.

Consequently, a music graph can never be exact - some people will find similarities inappropriate, to which others would agree. Strongly associated with this problem is the problem of the "ground truth" in music similarity [9]. What is the best quality measure once such a graph would exist? Where is a source which has undoubted similarities between artists or songs? In short, one can either rely on what people personally decide to be similar, or on machine algorithms which analyze the music as a sequence of frequencies. These approaches are described in more detail later.

1.3.1 Songs or artists?

One of the first questions to answer was if the graph should base on songs or artists. There were several relevant points which play a role in this decision:

- What has already been done? What problems have already been solved?
- Does it make sense to compute song-similarities? Or are songs of the same artist anyway alike?
- Would it be feasible (with respect to space and computational complexity) to create a song-graph? How many songs do exist in the world of music?

A graph on song-level would definitively be preferred with respect to applications. Possible users have songs on their computers, and these songs could be an input for an application. Furthermore, songs rather than artists are shared and downloaded over the Internet and not artists.

1.3.2 Find Data Source

In order to construct a graph of music in which similar songs or artists are interlinked, a notion of similarity has to be found. Of course, it is not possible to analyze each piece of music in the world in order to find out which ones are similar. Further this would only be the opinion of one person and possibly differ from others. The goal is therefore to find a source in the web which is freely accessible and relates different songs or artists in some sense.

1.3.3 Distance Measures

In order to create a reasonable application on top of the graph, the distance between two random nodes must be computed. This distance can be derived directly from the graph structure or via some distance labeling or embedding. A distance labeling would have the clear advantage of not being bound to a server. Computations could also be made distributed without knowledge of the whole graph.

1.4 Small-World Networks

For a graph to be a small-world-network, it should have three special features:

- A high clustering coefficient
- Sparseness
- Short average path length

If a graph has this properties, it experiences the "Small-World-Phenomenon". That means there exist short paths between any two nodes in the graph. Stanley Milgram first discovered this fact by using letters which had to be sent as near as possible to a target-person in the United States. He observed that there were small paths from any person to any other, and that the letters managed to get to their target in average 6 steps. The famous word "Six degrees of separation" was born.

Jon Kleinberg later founded the theory about the "Small-World-Phenomenon" [12] and explained the findings of Stanley Milgram with a theoretical background. Some networks are already empirically proven "small-world-networks", like the graph of film actors (collaborations as edges), the power grid of the western United States or the neural network of the worm caenorhabditis elegans [21].

A graph of songs or artists is also expected to be a small-world-network. In the world of music, clusterings exist for different genres or language regions. So it is expected to exhibit its properties of a high clustering-coefficient and short paths with few links.

1.5 Outline

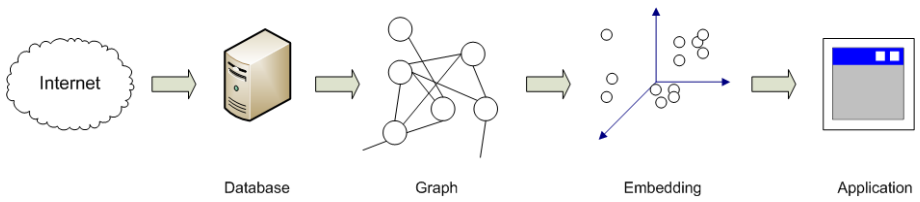

Figure 1.1: Overview of the steps taken in this thesis

In Figure 1.1, an overview over the whole work is given. The steps in this overview also correspond to the chapters in this thesis. Chapter 2 summarizes the work that has been done concerning music similarity, in research and applications. Chapter 3 deals with sources about music similarity available in the Internet and their crawling. Chapter 4 shows how a graph of songs can be constructed out of the data. In Chapter 5, the graph constructed before is embedded. Afterwards in Chapter 6, an application is designed and implemented which uses the embedding. At last, a summary over the whole thesis as well as some critical remarks and conclusions are given in Chapter 7.

2
Related Work

Since music is a very popular topic, many things have been done already, be it academic papers or commercial projects. Music similarity is expected to be a commercially usable feature. It can be used by music-vendors to sell more music, for example it is thinkable that it can somehow be predicted which kind of music will be popular in future. Some interesting work is presented below in more detail.

2.1 Research

Several studies have already been conducted regarding music similarity. Only an extract of them is mentioned here. There are three major strategies of how music similarity can be obtained: from metadata, from user-generated data or from the audio signal (audio-based, content-based).

Metadata

Music similarity can be generated from metadata, although metadata often is subjective. Furthermore it is hard to obtain it for large amounts of songs. Platt [17] constructed a graph of music from a metadata-database to compare different embedding algorithms for sparse graphs. Aucouturier and Pachet [1] generate playlists from a given metadata-database (17000 songs) satisfying constraints as duration, increasing tempo or genre closeness.

User-generated data

User-generated data (playlists, streams, user ratings) is also subjective, but large amounts of data are freely available. Of course each single user has his own perception, but it is hoped that in sum the obtained relations reflect the true similarity. Even if this is not the case, it reflects the opinion of the majority of users. Rasmussen, Gleich, Zhukov and Lang [5] embedded a graph of musical artists in three dimensions for visualization, using user-ratings

for artists. Ragno, Burges and Herley [19] built a graph of songs with 60499 nodes from authored streams (e. g. playlists of radio stations). They propose algorithms to generate playlist from a given seed-song.

Audio-based

Computing music similarity from the audio-signal is a purely objective measure. It is independent from metadata or the perception of individuals. On the other hand, the evaluation of the quality of such a measure again is subjective. Aucoutier and Pachet [2] propose an algorithm to compute audio-based similarity of music. Algorithms for playlist-generation from audio-based similarity are proposed by Logan [13].

Evaluation

The big problem of measuring the quality of music similarity is faced by Berenzweig, Logan, Ellis and Whitman [4] evaluated different similarity measures for music, from audio-based to metadata-based approaches, and faced the problem of "ground-truth" in artist similarity [9]. Pampalk, Dison and Widmer [16] compare and evaluate different audio-based similarity measures.

Playlist-Generation

Pampalk, Pohle and Widmer [7] generate playlists based on audio-based similarity dynamically, by observing the users skipping behavior. In another study [8] they apply travelling salesman algorithms on the whole graph. Having one big playlist, they propose to ease the browsing through a musical library with one input wheel. Van Gulik and Vignoli [20] present a visual playlist-generation from a given visualization of an artist-graph.

Contribution

In this thesis, music-similarity was obtained from users favorite tracks. A graph with more than 400000 nodes was created. Many algorithms proposed were not applicable to a graph of this size. The graph was embedded to an Euclidean space, to allow faster algorithms and the development of distributed algorithms. To evaluate the quality of the graph and the embedding, a new quality measure is introduced using a category-tree. It could be shown that IterativeLMDS improves the quality of the embedding compared to LMDS. Furthermore, some algorithms for playlist-generation out of an embedding are presented.

2.2 Applications

Pandora / Music Genome Project The "Music Genome Project" was created in January 2000, since then it goes on classifying music. Pandora is the Webradio-station, which uses the "Music Genome Project" to recommend new songs to the listener.

The name "Music Genome Project" is adapted from the famous slogan "Human Genome Project", which tries to split the human DNA into parts, analyze, interpret

and understand it. The same claim meets Pandora. Experts as well as algorithms analyze music using hundreds of attributes or "genes". The analysis of a song takes about 30 minutes and is partially done by experts. So the "human factor" is not fully discarded. But with this rather technical classification of music also unpopular music is presented to the listener. So it is a good source of finding new music which one could like, but otherwise never would have heard of.

Last.fm / Audioscrobbler Last.fm is a social networking platform for music-enthusiasts. It is based on "Audioscrobbler", which traces every track a registered user listens to on his PC and sends it to a server. This way, last.fm is able to show each user's statistics about music preferences and has a large database about the users favorite songs and their listening behavior. Concerning music similarity, last.fm provides a list of similar artists, which is computed out of the Audioscrobbler-data.

MusicIP MusicIP[1] developed an acoustic fingerprint of music. Songs can be identified by their fingerprint (regardless of file format or compression) using their free service "MusicDNS". The metadata returned by MusicDNS is used by MP3-Taggers like Musicbrainz' Picard[2].

MusicIP also created audio-based algorithms that identify similar music, based on mathematical analysis. Based on this algorithms, MusicIP sells so-called "playgrounds", with which a user can browse through his library on similarity links.

[1] http://www.musicip.com
[2] http://musicbrainz.org/doc/PicardTagger

3 Music-Information Sources

In the web, music is a very popular topic. Websites dealing with music and artists are widely spread. Nevertheless, structured and homogeneous information about the world of music is difficult to find. There are projects which give a system to the colorful world of music. Some of them are listed below. Unfortunately, not all of them make their work accessible for research issues. Thus, the list of possibilities shortens a lot. This chapter shows an evaluation of possible sources of information about music, and explains why last.fm seemed the most promising source. Afterwards it is described how the last.fm data was crawled and processed to later serve as basis for a graph of songs.

3.1 Evaluation of Possible Sources

3.1.1 Overview

Table 3.1 gives a broad overview over the evaluated sources. The first column shows if a graph on song-level can be constructed, the second column indicates how complete the data is and the third shows if it is freely available.

	song-level	completeness	availability
Allmusic.com	√ √	√	
Wikipedia	√	√	√ √ √
Pandora (Music Genome Project)	√ √ √	√ √	
Search Engines	√	√ √ √	√ √ √
Last.fm	√ √	√ √ √	√ √

Table 3.1: Overview over the evaluated sources.

3.1.2 Allmusic.com

Allmusic.com is a popular music-database. An expert-community is maintaining the information which is stored about artists, albums and songs. Also, concerning similarity information, they provide some interesting information.

Concerning artist-relationships, the allmusic-database contains for some artists a short list of similar artists. But this information is quite fragmentary. For some artists no similarity is stated and for others only very few. Nevertheless allmusic.com seems to be the best hand-made source which is freely accessible on the web. For similarity on the granularity of songs, allmusic.com provides style-, mood- and genre-information of the (about 100) most popular songs. This amount of songs is of course far too small to construct a reasonable graph of songs, but using this style-information, a similarity measure can be computed [17].

The core business of allmusic.com is the licensing of their information to companies like microsoft or amazon. These companies use the data to enhance their applications. Unfortunately allmusic.com is not crawlable. After some number of requests the crawler gets banned. As a consequence, this source of meta-information can not be used for constructing a music-graph.

3.1.3 Wikipedia

Wikipedia in its different language versions is holding a lot of information about music. For lots of artists, style-information and a descriptive text is given. Since Wikipedia is a wiki and therefore 'hand-edited' by the community, the pages of the different artists are individual. The crawling of these different-looking pages would require complex parsing and is not feasible within this master thesis.

Another weakness of Wikipedia as information-source for music is its incompleteness. It is in the nature of a wiki, that unpopular and rather unknown topics are not perfectly covered. Also information at song-level is hardly available, it exists only for very popular tracks as those of the Beatles.

3.1.4 Pandora / Music Genome Project

Pandora is a personalized Web-radio[1]. As starting point it gets the favorite artist or song. From then on, it plays music that should be similar. Behind Pandora lies the 'Music Genome Project'; the name is derived from the famous 'Human Genome Project'. This project was founded in January 2000, and since then analyzes songs. It has the ambitious goal to divide every piece of music into 'genes'. Therefore it uses expert-opinion and acoustic algorithms.

The music genome project analyzed over 400000 songs from more than 20000 artists (July, 2006, source: Wikipedia). This database is the core of the company so it is comprehensible that Pandora does not make information accessible, and does not make available a data-sample for reseach issues. However, on their website, Pandora offers a "Background Search"-function, which indicates a sample of 6 similar songs for each song in their database.

[1] http://www.pandora.com

But this small and incompletely accessible information, together with the uncertainty if crawling would be legal, discourages from using Pandora as source for a music-graph

3.1.5 Search Engines

Search engines are a known source for extracting social networks. E. g. in the development of POLYPHONET [14], search engines are extensively used. It is described how a social network of the academic society is constructed using Google.

The way relations are built using search engines is as follows:

1. Number of results n_i for query 'keyword i'
2. Number of results n_j for query 'keyword j'
3. Number of results $n_{i,j}$ for query 'keyword i AND keyword j'

With this triple $\{n_i, n_j, n_{i,j}\}$, the relatedness of two keywords is computed using known formulas (see Section 4.1.1).

For music (be it artists or songs), the information search-engines provide seems to be inadequate. Web-pages holding two songs are rather chart-lists or discographies than lists of similar songs. So a co-occurrence of two songs on a web-page has nothing to do with similarity in many cases. For this reason, search engines like Google were not taken into account in the evaluation of possible data-sources for music similarity information.

3.2 Last.fm / Audioscrobbler

Last.fm is a social network, where registered users have a profile and a list of friends (see Figure 3.1). The 50 most played tracks ("top-tracks") of each user can be retrieved. Songs which appear together on such a list are in most cases related, so they can be assumed to be similar in some sense. However, such two songs do not have to be the most similar ones. How similarity is constructed out of these top-tracks is described in Chapter 4 in detail.

Audioscrobbler (the project behind last.fm) provides some REST-Webservices[2] which are freely available. A REST-Webservice encodes the parameters in the url (like http://www.zipcode.com/ws/country?cityname=xyz) and gives the information back in xml-format or plaintext. The advantage of this approach is the fact that it works the same way as the world-wide-web. The request does not have to be encoded in a special format (like SOAP) and the webservice can therefore be consumed by simple clients (like websites with javascript).

The only constraint the Audioscrobbler Webservice has is that it is not allowed to do more than one request per second. This constraint was not critical to the project, since one request per second means 86400 requests per day. This number of requests is high enough to get a useable amount of data in a reasonable time (the thesis took six months). There

[2] http://www.audioscrobbler.net/data/webservices/

are other quite similar services (musicmobs.com, upto11.net), but last.fm is the most popular and therefore the most adequate for the task of constructing a graph of the world of music.

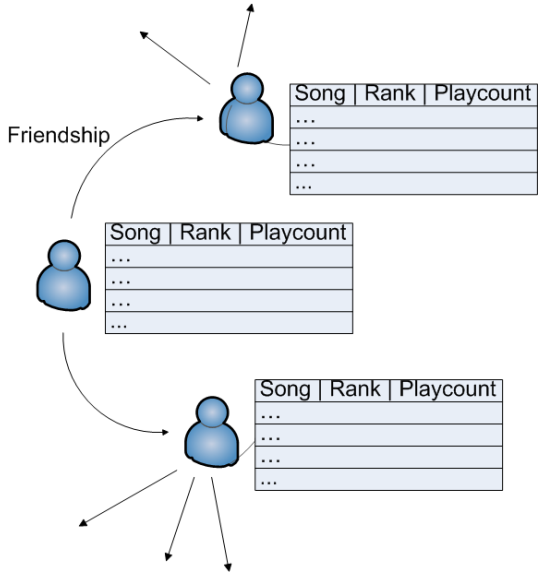

Figure 3.1: Simplified model of the last.fm-community. A friendship-network, where every user has its music profile

With respect to possible applications, the opinion and the feeling of the "end-users" themselves seemed a good source. The alternative would be expert opinions or some sort of acoustic comparisons (which was out of scope). Last.fm is always up-to-date and independent from experts, which are of course also subjective in their findings. Another nice characteristic of the Audioscrobbler-data is that it allows to compute both artist- and song-similarity. Considering this advantages and the disadvantages of the other sources, we decided to work with last.fm/Audioscrobbler.

3.2.1 Retrieving the Data

Using the REST-Webservice of last.fm, the 50 most-heard tracks of each user can be retrieved. Unfortunately, last.fm has the policy of not giving away large amounts of data although it is freely accessible in small pieces. So a sample of data has to be fetched by a automated crawler. Since it is allowed to state one request per second, the amount of data that can be fetched is limited by time, but sufficient to get a quite large sample (see Table 3.3). Our crawler has two basic functions (see Figure 3.2): expanding users and crawling top-tracks.

A mysql-database is used in this project for storage. The data-model corresponds to the data in last.fm (see Figure 3.3). The tables for users and friendships are filled when expanding users. The tables for songs, artists and top-tracks (tblHas_song) are filled up when crawling the users' top-tracks.

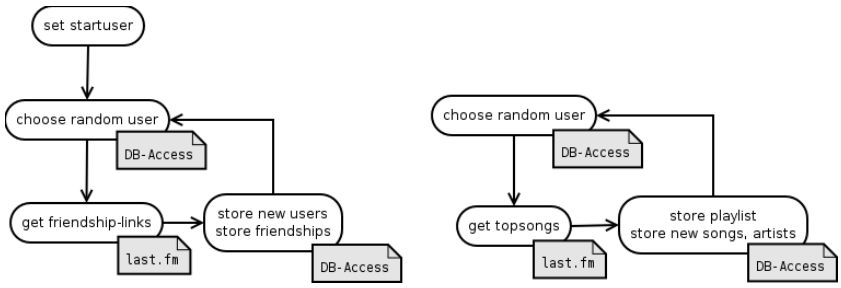

Figure 3.2: Expanding users (left) and crawling the top-tracks (right)

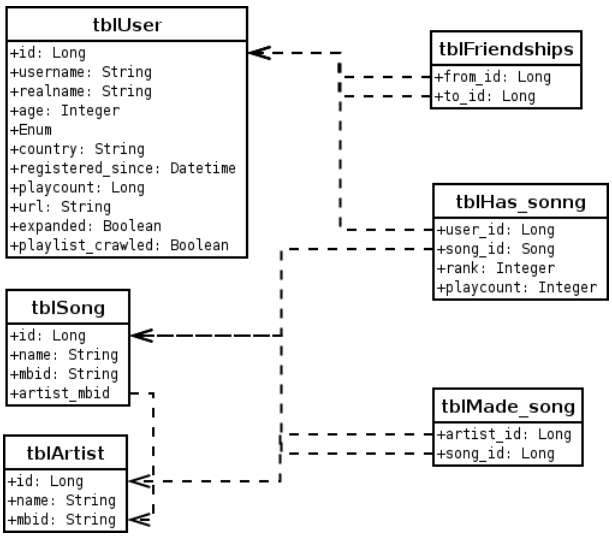

Figure 3.3: Relational tables for storing the data from last.fm

Possible Improvements

Expand users and get the top-tracks of users which are on the "edge" of the network. Take the user with the least friends in the database to be the next to be expanded. Like that, the crawled sample of users would spread more "homogenous" over the whole friendship-network. If we assume that friends tend to listen to similar styles of music, the graph of songs would also get more homogenous in the world of music.

3.2.2 Cleaning Process

The data of last.fm is in fact what the users provide. That means the names of the songs and the titles may be misspelled or even incorrect. Table 3.2 shows an exemplary extract of the data to illustrate the problem. To reconstruct the incorrect data is in general not possible since it is not known what song was really played by the user.

id	name	musicbrainz.org-id (mbid)
137	The Beatles	b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d
34012	The Beatles (Love Album)	
89826	The Beatles @ .com	
125937	The Beatles-LOVE 2006	
133814	The Beatles - No Reply	
135968	The Beatles'	
145964	The Beatles: Paul Mccartney	
149681	The Beatles @ .nl	
155591	Meet the beatles	
160674	Beatles, Let it be	
200856	1025 Beatles	

Table 3.2: Extract of table "tblArtists". Many different versions of the same artist exist, but only one has a musicbrainz.org-id (see Section 3.2.2) attached.

For simplicity, no data is altered in the cleaning process, entries are only accepted or deleted. If an artist's name is not found in the musicbrainz-database[3], the artist is deleted with all his songs. Also all top-tracks-entries containing these songs are deleted.

Musicbrainz

Musicbrainz is a large online-community which is committed to bring order into the world of music. They have a database with over 300000 artists and 5000000 tracks (April 2007). Each artist has a unique id called MBID and a lot of information stored. Associated with the artists are albums and tracks. The tracks have also an unique MBID and an acoustic fingerprint. This acoustic fingerprint, called PUID, is proprietary by Music IP (see Section 2.2), which collaborates with musicbrainz.

[3] http://www.musicbrainz.org

Musicbrainz makes all its data available in a REST-Webservice. This webservice is used to clean the last.fm-data as described in Section 3.2.2.

Data Statistics

The cleaning process results in the following data, which was used to compute the graph (see Table 3.3).

Total users	290'148
Distinct songs	1'570'519
Distinct artists	91'470
Avg songs/artist	17.2
Avg toptracks/song	9.24

Table 3.3: Data statistics after cleaning

The songs/artist-ratio shows how many songs an artist has on average in the crawled data. The toptracks/song-ratio on the other hand shows how many times a song appears in the playlists on average. The second ratio is important in order to create a meaningful graph. If a song only appears once in all top-tracks, there cannot be said much about the relatedness of this song to other ones. Of course, the more data is at hand, the more the songs occur and the better the quality of the graph gets.

3.3 Handling of Large Tables

In the course of this master thesis, a lot of data was gathered which had to be processed. This data was all stored in different MySQL-databases. To handle the large tables, there are some strategies and "tricks" which helped a lot. Most important and the goal of all following points - reduce database-accesses as much as possible. In this project, Java and JDBC were used, but this recommendation typically also holds in other configurations.

Batch-Updates When a lot of updates, inserts or deletes are required, batch updates should be made. This way, the database is accessed only once instead of hundreds of times. This reduces the database-accesses and speeds up the application.

Streaming result sets When doing a request with JDBC, the whole resulting table is stored in memory before it can be accessed. This fact can lead your application to run out of memory. To process the rows anyway, a streaming result set can be used. Only the actual row is loaded into memory at a time. The drawback of the streaming result set is that no other query can be executed until the streaming result set is closed.

Subqueries When the database allows subqueries (MySQL does so from at least version 5), they are faster than to process the inner query in your application code.

Indices Indices should be used very carefully. They speed up select-statemens, but slow down updates and inserts significantly (especially when the indices are organized as

B-Tree). The index has to be maintained with each change of the table. So when tables get large and frequent inserts are needed, every index that is not essential should be omitted.

4 Graph Construction

There are different possibilities how to construct a graph of songs out of the described data (see Chapter 3). Depending on the strategy, other links between songs and other weights of the links will be generated. Because of this large variety of possibilities, it is inalienable to have a certain "ground-truth" to which the constructed graph can be compared.

4.1 Pairwise Similarity

Having the top-tracks of the last.fm-users, the first problem to solve is the method how to relate two songs. The basis on which the whole graph is founded on is the following assumption:

Playlists contain similar music

and therefore

The more top-tracks contain both together, the more similar the songs are (assuming all other parameters stay constant).

There are some difficulties to consider using this heuristic:

- Two very popular songs are more likely to be together in the top-tracks of a user, so popular songs should be "punished" somehow.
- Songs which occur only once are difficult to treat. They have 49 neighbors, namely the songs of the top-tracks-list in which they occur. A similariy calculated from this scenario is clearly doubtful.

4.1.1 Similarity Indices

To describe the different coefficients, a few simple terms are introduced:

N Number of playlists

n_i Number of occurrences of th song with id=i

$n_{i,j}$ Number of co-occurrences of the songs with the ids i and j

$users(s_i, s_j)$ Set of users which have the two songs with ids i and j in their top-tracks

A co-occurrence of two songs is the event that these two songs occur in the same top-tracks-list. That means the two specified songs are listened to by a person. We restricted the pairwise similarity between two songs to depend only on the triple $\{n_i, n_j, n_{i,j}\}$. Having the data from last.fm (see Figure 3.1), it would also be possible to consider other information. Further ideas are explained in Section 4.1.2.

Adapted to the environment with the songs and the top-tracks-lists, some similarity measures from the information retrieval community are known from literature[14].

Co-occurrences:	$n_{i,j}$
Cosine Coefficient:	$\frac{n_{i,j}}{\sqrt{n_i} * \sqrt{n_j}}$
Dice Coefficient:	$\frac{2 * n_{i,j}}{n_i + n_j}$
Overlap Coefficient:	$\frac{n_{i,j}}{min(n_i, n_j)}$
Jaccard (Tanimoto) Coefficient:	$\frac{n_{i,j}}{n_i + n_j - n_{i,j}}$

Dice and Jaccard are equivalent with respect to ordering, since $dice = \frac{2 * jaccard}{jaccard+1}$. For the ranking of the similar songs it makes therefore no difference whether to use Dice or Jaccard. But the Jaccard-coefficient is a little more "rigorous" for less-related entities. E. g. if the similarity computed with Dice is 0.8, Jaccard returns only $\frac{2}{3}$.

Own Approaches

triple-probability This similarity coefficient measures the probability that a triple $\{n_i, n_j, n_{i,j}\}$ would occur in N random top-tracks-lists.

$P[s_i] = \frac{n_i}{N}$
$P_{triple} = ((\frac{n_i}{N})^{n_{i,j}} * (1 - \frac{n_i}{N})^{n_j - n_{i,j}}) * Permutation(n_j, n_{i,j}, n_j - n_{i,j})$
$weight = -\log P_{triple}$

Explanation: Given $users(s_j)$ and $users(s_i, s_j)$. $n_{i,j}$ of the occurrences must exactly match the $n_{i,j}$ playlists in $playlists(s_i, s_j)$. The probability for one of these events is $\frac{n_i}{N}$. At the same time, $n_j - n_{i,j}$ times the occurrence of s_i must not match the top-tracks-lists which s_j is in. This gives the probability of this triple to occur in one special sequence. To get the overall probability this term has to be multiplied with the permutation of all possible sequences.

co-probability The co-probability relates the natural probability that two songs co-occur (based on their overall occurrence) to the actual co-occurrence $n_{i,j}$.

$P[s_i] = \frac{n_i}{N}$
$P[s_i, s_j] = \frac{n_{i,j}}{N}$
$weight = \log \frac{P[s_i,s_j]}{P[s_i]*P[s_j]} = \log \frac{\frac{n_{i,j}}{N}}{\frac{n_i}{N}*\frac{n_j}{N}} \sim \frac{n_{i,j}}{n_i*n_j}$

To emphasize the value $n_{i,j}$, it can also be squared. We then call the coefficient $co-probability^2$.

$$weight = \log \frac{\frac{{n_{i,j}}^2}{N}}{\frac{n_i}{N}*\frac{n_j}{N}}$$

The weight is then equivalent (with respect to ordering) to the cosine index, since $co-probability^2 = log(N * cosine)$. The $co-probability^2$ is however better for the purpose of song-similarity, because it favors low similarities, whereas the cosine-coefficient produces very low weights for most cases.

4.1.2 Considering Rank- and Friendship-Information

The last.fm/Audioscrobbler-data also contains friendship information. This information made it possible to retrieve a random set of top-tracks-lists by randomly walking through the friendship links. Because only a subset of all friendship-links are accessible via the web-service at the time of writing, it was not possible to get the complete friendship-graph from last.fm. So we could only get a subgraph of the real friendship network.

Assumption: The less connected the users(s_i, s_j) are, the more connected are the two songs.

Explanation: Two friends are more likely two share the same songs, even if they are not highly similar. Thus it seems to be an even stronger indicator for similariy if two unrelated persons share two songs in their top-tracks. This influence is however assumed to be marginal and therefore not further explored in this project.

The top-tracks of the users contain also rank- and playcount-information (see Figure 3.1). The playcount is the number the user listened to the specific song. Also this data can be taken into consideration:

Assumption: The higher the playcount-numbers are, the more important is a specific co-occurrence

Explanation: If a user listens only sporadic to music, his opinion is less important to a music-enthusiast. If a song is only heard twice in a month, that obviously doesn't mean a lot to the similarity inside this top-tracks-list.

4.2 Directed or Undirected?

When talking about artists, one would clearly agree that the similarity relation must not be mutual. E. g. a former artist that influenced a new one (e. g. The Beatles and Oasis) would not be called similar. But in the other direction the new artist is similar to the artist that influenced him. The notions of time and style influence are mixed with the notion of technical similarity.

But between songs, this influence is less natural. Songs can be compared at a more technical level and the social components are less present. Two songs can be similar for different reasons (instruments, lyrics, mood, melody) than artists. Thus it is reasonable to have no direction on the edges between songs.

4.3 Construction of Edges

Out of the described pairwise similarities, a graph $G = (V, E)$ with n vertices and m edges has to be computed. In our graph, the vertices represent songs. The pairwise similarity measures gives a weight for each edge in the graph. Assuming that every song appears only once and there are 1 million nodes, the following simple calculations can be made:

$$m = 49 * n = 49millions,$$

since every vertex has 49 neighbors (the songs in the same top-tracks-list). Having an average occurrence of 7, this number increases:

$$m = 7 * 50 * n = 350millions$$

With the ongoing crawling and adding of top-tracks-lists, the number of edges gets even larger. To handle the graph and perform algorithms on it, it is necessary to sparsen it, i. e. to remove some edges. To achieve this, two major strategies can be found.

4.3.1 Different Sparsening-Strategies

Fixed Number of Edges (rank-based)

Idea: To limit the total number of edges in the graph, the number of outgoing edges per node can be limited. For each vertex, the top-weighted x edges are stored. This strategy has advantages and drawbacks

- Every vertex has neighbors
- If x is set large enough, the graph is connected with high probability
- “Weak” edges will be in the graph, whereas “stronger” edges are omitted

Threshold (weight-based)

Another strategy to find the edges which are worth storing is to fix a threshold α, and to omit all edges with $weight < \alpha$. This approach is more intuitive, since the weakest edges

are omitted first. That means that only edges with a certain amount of similarity stay in the graph.

- A threshold has to be found such that the graph is connected
- The "strongest" edges stay

When counting the number of nodes in the largest connected componenet with increasing threshold, an optimal value can be found (see Figure 4.1). The graph should be connected (the largest connected component should desirably be as large as the graph itself), but there should be as few edges as possible. So the right point for the threshold is where the curve starts to flatten. That means there are not much nodes added when further increasing the threshold, but mainly edges.

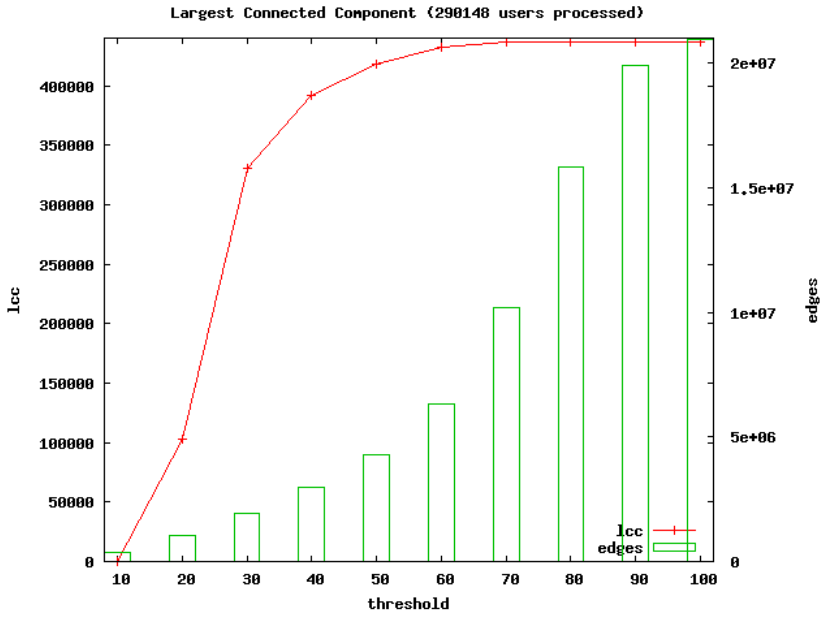

Figure 4.1: Largest Connected Component for different thresholds after 290148 users processed. The pairwise similarity measure used in this test is $co - probability^2$.

4.4 Quality Measures

4.4.1 "Ground-Truth"

Finding ground truth in music-similarity is a known problem. For artist-similarity, [9] already evaluated some sources to serve as ground-truth and made different datasets available. But to be reasonable for a song-graph, the ground-truth also has to apply on the level of songs. Otherwise the song-similarity could not be checked and it would make no sense at all to construct a graph at the granularity of songs. Therefore, we had to find a reasonable source to compare the results ourselves.

Since music is not an exact topic and lacks a mathematical model, similarity depends on the human observer. Nevertheless, something is needed to measure the quality of the links of the created graph. There are at least three possibilities to do this, of which we chose the easiest with respect to time and budget constraints.

Ask People

The first solution which comes to mind is certainly a sort of poll or user evaluation. The generated similarities are presented to ordinary people which then give feedback whether this similarity is good or not. Because the outcome of such an evaluation depends greatly on the persons that are asked, a certain bias in the user's opinion is very probable. This problem can be countered with more test-persons.

It seemed difficult to generate a big and heterogeneous community of test-users in short time. The quality measure with user evaluation was not considered.

Other User-Driven Recommendation Systems

There exist recommendation systems on the internet, and they are also accessible (last.fm, upto11.net, musicbrainz.org, musicmobs.com). But all of them just provide artist similarity. Of course this can also be used, to give an overall impression over the sense of the generated links. If a link exists between two songs whose artists are recommended mutually by the recommendation systems, the link can be appropriate. But it doesn't help getting the quality on the level of songs.

Because there was no freely accessible song-level recommendation system, this quality measure was also out of question.

Expert Systems

Another possibility to measure the quality of the constructed links is to compare them to an expert-opinion. If there would exist a database of links between songs edited by some experts, that could serve as a sort of "ground-truth". Unfortunately, such data is hard to find, but allmusic.com provides a kind of similarity between songs feasible to crawl, which is described in the next section.

4.4.2 Style-Information as Ground-Truth for Song-Similarity

Allmusic[1] is surely one of the largest music-database which exist on the internet. [9] used allmusic.com in their studies as ground-truth for artist-similarity. However, it can also serve as source for song-similarity. Allmusic provides (among other things) a hierarchical tree of music styles. Each of these styles has a list of "main-songs" (see Figure 4.2), songs which are typical and popular for a certain style. These lists are hand-made by the community.

These "main-songs"-lists can be used to measure the quality of the computed similarities. Songs which appear on one of this lists should be similar to the other ones, or at least to the

[1] http://www.allmusic.com

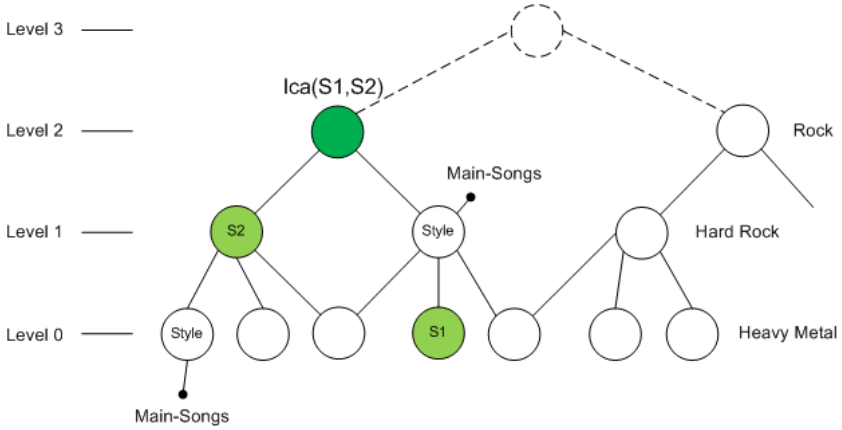

Figure 4.2: Style-graph of allmusic.com. The distance of two styles is the level of the least common ancestor. E. g. the two highlited styles have distance 2.

ones of the parent style or of sibling styles. The most natural distance function for two songs s_i and s_j or styles s_a and s_b in this graph is based on their lca (least common ancestor):

$$styleDistance(s_i, s_j) = level(lca(styleOf(s_i), styleOf(s_j)))$$
$$styleDistance(s_a, s_b) = level(lca(s_a, s_b))$$

The levels are according to Figure 4.2. If two connected songs are within distance 0, they are from the same style. If they have distance 3, they are only connected by the artificial supernode, that means they are not related at all. Any distance function which is used with the music-graph can now be compared to the distances given by the style-graph. If two songs have distance 3 in the style-graph, they should be far away in a graph representation. This fact is used to measure the quality of the link creation, the graph construction and the embedding.

For the implementation of the quality measure, the styles and their main-songs have to be crawled from Allmusic.com. To avoid being banned, only one request per 10 seconds was made. Of course, not all of the songs in the allmusic.com-main-songs were found in the crawled sample of last.fm, since there are very special styles in allmusic.com like “Traditional Middle Eastern Folk”. And some styles do not even have main-songs assigned.

The style-graph is not exactly a tree. Styles can have multiple parents (as in Figure 4.3). Nevertheless it has no cycles, since the edges are directed from child to parent, and only point from a lower level to higher one (i. e. the graph is a special directed acyclic graph). So the least common ancestor, which is used as distance measure, can be defined exactly.

styles	735
styles with assigned main-songs	610
distinct songs	10534
distinct songs found in sample	6654

Their occurrence in the "toplevel-styles" (main categories like rock, jazz or blues) are as follows. The same song can appear several times, so the sum of all songs is beyond the number of distinct songs.

Rock	6602
Jazz	908
R & B	1695
Rap	170
Country	1198
Blues	1004
World	831
Electronica	1079

4.4.3 Results

For our sample of songs, the links have been created using the different pairwise similarity measures (see Section 4.3.1). Afterwards, the distance can be computed for all edges.

1. Construct graph using a specific strategy (pairwise similarity coefficient, threshold or fixed number of links)
2. For each edge, compute the distance

Because edges which base on only one co-occurrence are rather random, these edges are omitted for the quality measure. In Figure 4.3, the results of a quality-measure is shown. Up to almost 90 percent, the constructed edges are within distance 2, that means there are only 10 percent of the edges whose styles are not connected according to allmusic. The $cosine$- and the $co - probabilty^2$-index and the $triple - probability$-index (see Section 4.1) seem to be the best indices for constructing similarity out of lists with persons favorite songs.

Similarities can be constructed out of the weights (which are dissimilarities) easily, e.g. by just subtracting from 1. In Figure 4.4, the weights are mapped to similarities between 0 and 100 (0 means perfectly similar). It is obvious that using the indices $jaccard$, $dice$, $cosine$ or $overlap$ results in weights that are mostly between 90 and 100. Theses indices do not differentiate the different strengths of similarities in our data. The indices $co - probabilty$, $co - probability^2$ and $triple - probability$ show more balanced distributions. Therefore, $co - probability^2$ is chosen to construct the links of the graph.

4.5 Implementation

The whole project was implemented in Java. As database connection, MySQL together with an according JDBC-Driver was used. For the construction of the edges, an incremental approach was realized (see Figure 4.5. Each song and each edge has a counter which is incremented whenever it occurs.

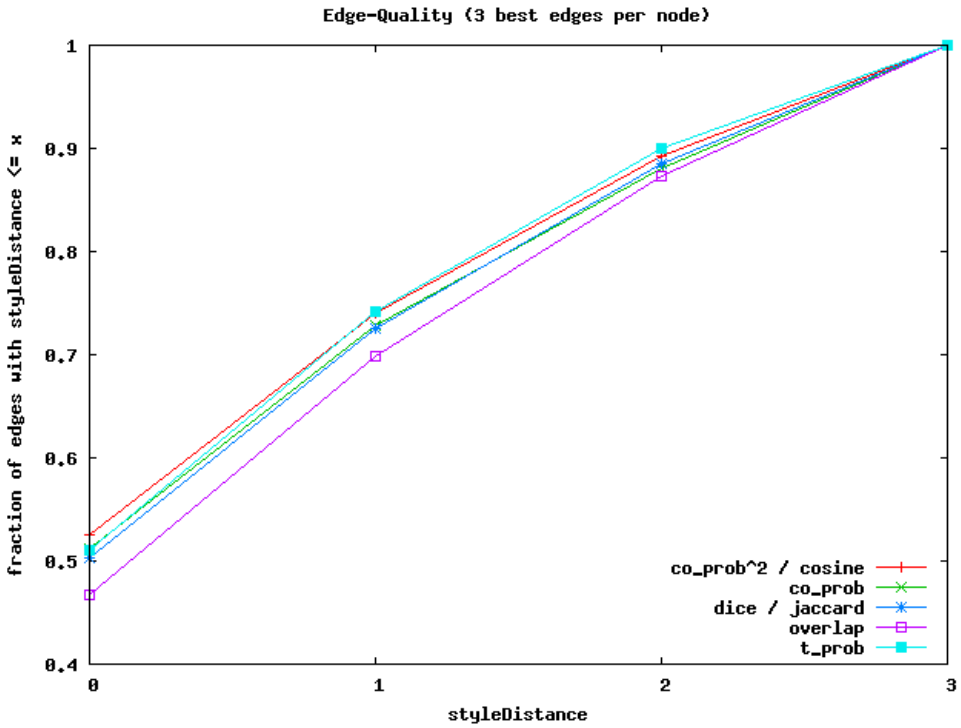

Figure 4.3: Quality measure for fixed 3 edges per vertex.The value at position 0 is the fraction of edges which connect nodes from the same style. The value at position 2 is the fraction of edges which connect nodes from the same top-category (Rock, Jazz, etc.), including the edges of position 0 and 1 (a subtree in Figure 4.2). Thus, the higher the curve lies, the better.

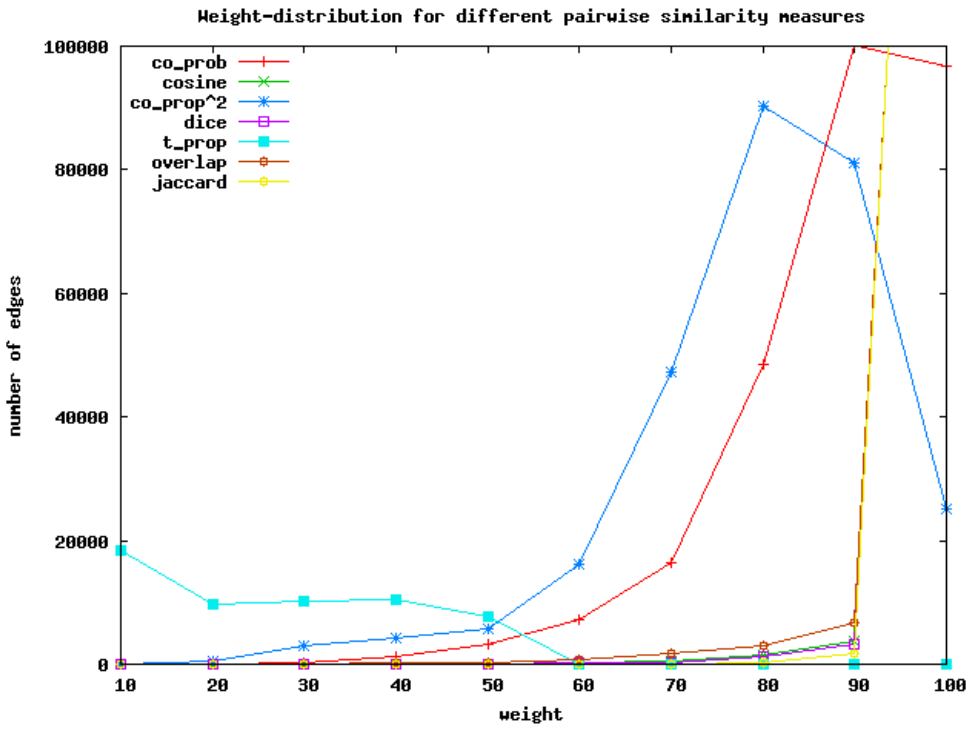

Figure 4.4: Weight-distribution of the edges constructed with the different pairwise similarity measures. The value at position 10 is the number of edges with weight between 0 and 10.

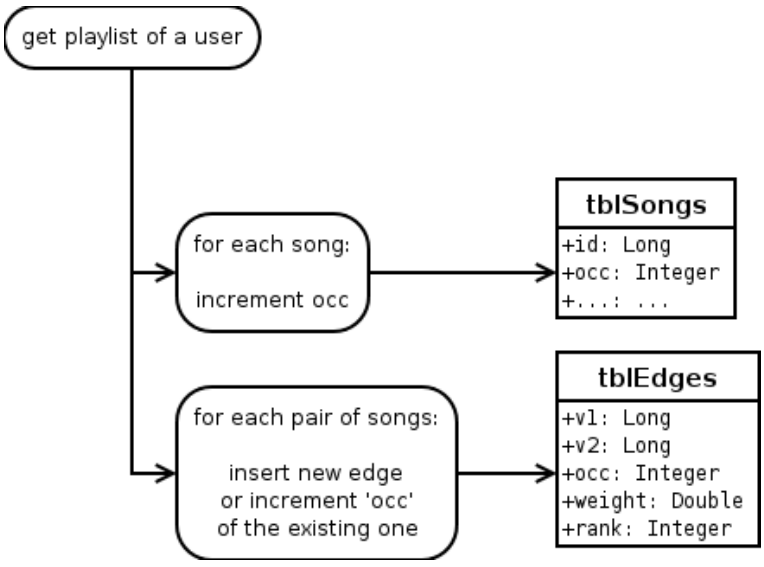

Figure 4.5: Processing a user

Each user is processed alone. Each song and each pair of songs (edge) in a user's top-tracks-list is incremented. After this computation, a user can be forgotten, and new users can incrementally be added to improve the quality of the links and to add new links. This approach is very natural and makes it easy to expand the data.

Complexity: $O(u * p^2)$ where u denotes the number of users and p the number of top-tracks-entries per user.
Time: Approximately 8s/user (most of the time used for updating edges)

4.6 Properties of the Graph

According to Figure 4.1 it is optimal to set the threshold to 60. Ignoring all edges with distances higher than 60, the following graph is built:

Nodes n	430'000
Edges m	6'300'000
Clustering coefficient [2]	0.5152826376728573
Diameter d (weighted)	370 ¡ d ¡ 523
Average path length	194.18

The high clustering coefficient and the low average degree agree to the properties of a small-world-network, as Watts and Strogatz [21] explained. This properties lead to a small diameter and small path lengths, which is the most significant feature of small-world-networks, as they gain their names from it. The graph of songs constructed out of people's most heard songs therefore joins the group of small-world-networks.

The degree-distribution of the graph is approximately power-law (see Figure 4.6). This degree-distribution also occurs in web-graphs [3].

[2] According to paper of Watts Strogatz [21]

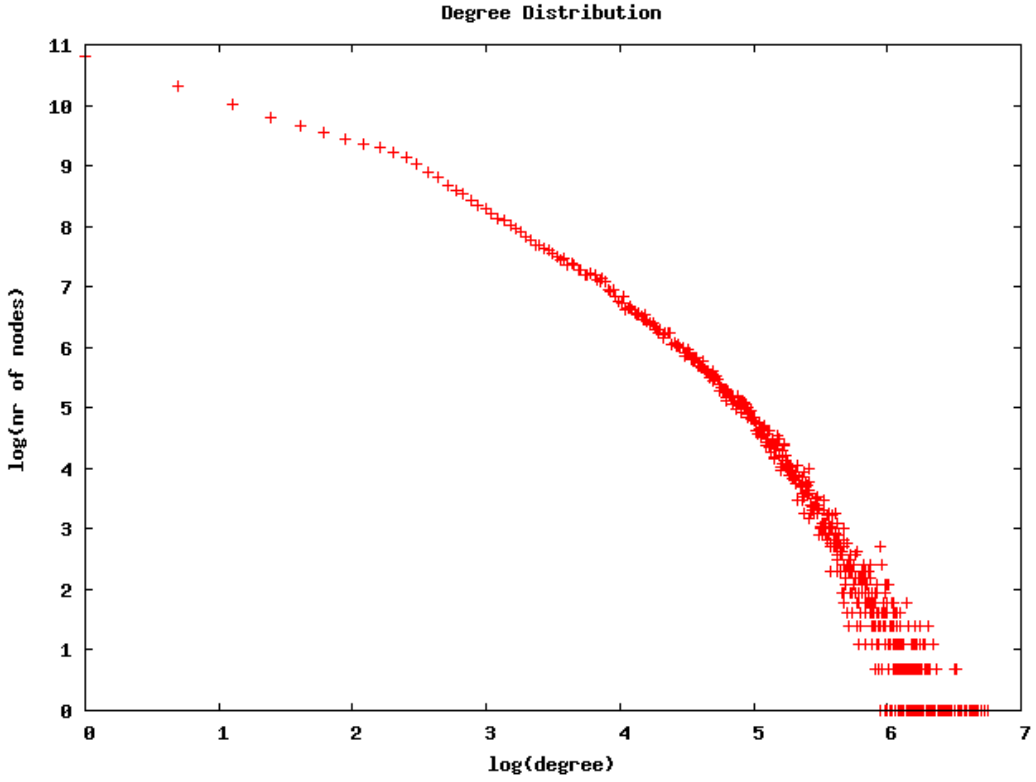

Figure 4.6: Degree-distribution of the music-graph. The distribution is approximately power-law.

5 Embedding

Given a graph $G = (V, E)$ with n vertices and m edges. An embedding of G assigns to each node in V a point in the Euclidean space $\mathbb{R}^d$, where d is the number of dimensions of the Euclidean space.

For the Music-Graph, the advantages of an embedding are obvious. A lot of algorithms such as computing the distance between two songs are drastically simplified. Moreover, an embedding makes many applications which are based on the Music-Graph independent of a central server, the application gets distributed (see Figure 5.1). Especially mobile-applications, which typically have a slow, and often not permanent connection to the Internet, this would be a significant feature.

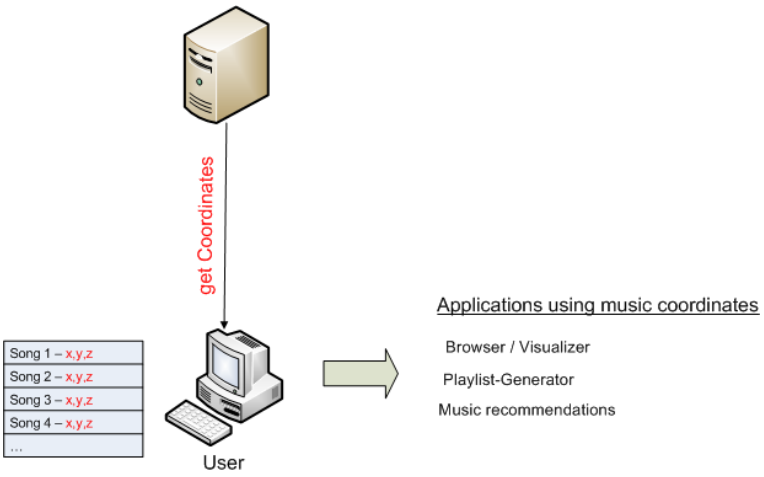

Figure 5.1: Sketch of applications with an embedding of a music-graph

5.1 Embedding of Large Graphs

When dealing with large graphs, a lot of embedding and visualizing algorithms are too complex (quadratic complexity is infeasible for a large number of nodes). Memory or computation time limits are reached. The general idea to break through this limits is to split or filter the graph in some way. The graph is divided in subgraphs and then embedded one after the other. This idea is applied in MIS-Filtering and Landmark MDS, which are able to deal with large graphs.

All-pair-shortest-path	$O(n^2 * \log n + m * n)$	Johnson
Single-source-shortest-path	$O(m + n * \log n)$	Dijkstra
Classical MDS	$O(d * n^2)$	
Landmark MDS [6]	$O(n * l * d + l^3)$	
Force-directed embeddings	$O(n^2 + m)$	Fruchterman-Reingold [10]
MIS-filtering [11]	$O(n * \log maxdegree^2)$	

Table 5.1: Complexities for some graph-algorithms. n denotes the number of vertices, m the number of edges, d the dimensions of the embedding and l the number of landmarks in Landmark MDS

Force-directed embeddings Also called spring-embeddings. These methods are not able to embed large graphs, they are only convenient for some hundreds of vertices (see [11]).

MDS, Landmark MDS There exist different versions of MDS (Classical MDS, FastMap, MetricMap, Landmark MDS). MDS works with eigenvalue-decomposition of the adjacency matrix of a graph. Platt [18] showed that Landmark MDS outperforms the other variants of MDS.

MIS-Filtering The graph to embed is first split into vertex sets which are maximum independent sets. These sets are sequentially embedded with a local force-directed method. The advantage of this algorithm is that no clustering is needed, the hierarchical splitting of the graph is according to the underlying graph structure. [11]

5.2 Embedding with LMDS

To embed the Music-Graph, we decided to use Landmark MDS. Landmark MDS first embeds a set of landmarks with classical MDS and afterwards positions every other node according to the distances to the landmarks. The algorithm assures to bring results with low computation complexity, since its complexity is controllable by the input parameters (landmarks, dimensions).

5.2.1 Choosing Landmarks

The set of landmarks has a significant effect on the outcome of LMDS, so it is important to think about possible ways of choosing them and which approach is best suited.

Random The landmarks are chosen at random.

Max-degree The nodes with the highest degree are the landmarks.

Max-min Proposed by de Silva and Tenenbaum [6]. Starting with a random node, the next landmark is always the node with the maximum minimal distance to the set of landmarks.

Max-degree favourises the most popular nodes (in terms of a small-world-graph). This can lead to bad embeddings since the popular nodes may all be in the same part of the graph (e.g. US web-pages in a web-graph). Random landmarks lead to good results when the number of them is high enough. But this worsens the performance a lot, since for every landmarks a SSSP-tree[1] has to be computed (see Table 5.1). Max-min gives also reasonable embeddings with small numbers of landmarks, and it is guaranteed that also unpopular parts of the graph (like classical music in our graph) have near landmarks. Otherwise whole groups of nodes can have the same distance to all landmarks and would therefore be embedded in a single point, which is not desirable.

5.2.2 Right Number of Dimensions

Besides the landmarks, the number of dimensions determines the outcome of the embedding. The determination of the number of dimensions is a trade-off between quality of the embedding and the space needed.

To measure the quality between the different dimensionalities, different playlists are generated (using Algorithm 4). The smoothness is computed as the number of re-occurring style-transitions in the playlist (e. g. if a playlist starts with a country-song followed by a jazz-song and country-song, this would count as one re-occurring style). As a playlist should smoothly lead from one style to another without many jumps, this serves as a good quality measure.

As Figure 5.2 shows, the smoothness of the playlists increases while incrementing the number of dimensions. 8 dimensions is the lowest number of dimensions where the outcome is still reasonable. This is the optimal, because the number of dimensions should be held as small as possible (because of space and computation complexity).

5.2.3 Quality Measures

An embedding has different parameters (number of landmarks, methods to choose landmarks, dimensions, sparsening of the edges). To find the best suited parameters, it is important to have a notion of quality. One can distinguish between structural and semantic quality measures. Structural means without knowledge of the content of a vertex (e. g. without knowing what songs lies behind the vertices). A semantic quality measure on the other hand can take into account this information.

Edge-Distortion A well-known structural quality measure of embeddings is the so-called distortion. The distortion d compares the distance of two vertices in the embedding and in the original graph. Because the full distortion is too complex (it would require

[1] Shortest paths from all nodes to a single source-node.

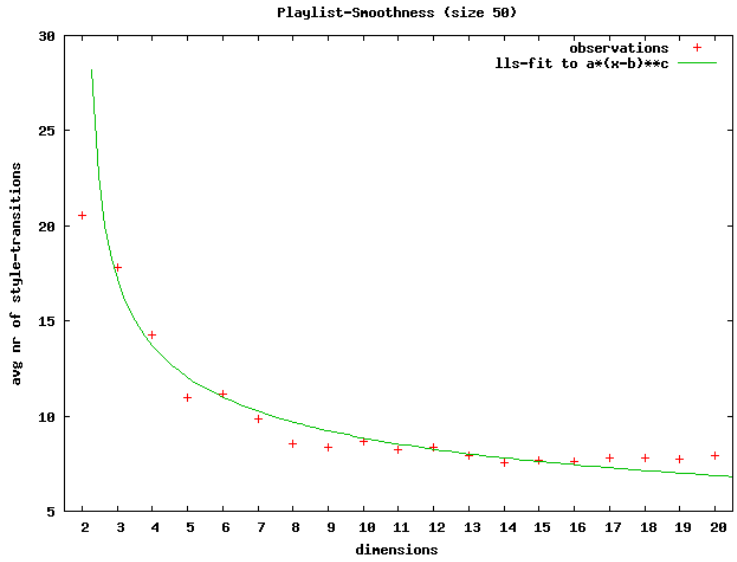

Figure 5.2: Playlist-Smoothness for playlists of size 50

a all-pairs-shortest-path), only the distortion of the edges that are present in the graph is considered. Formally:

$$d = \frac{1}{m} * \sum_{\{u,v\} \in E} max(\frac{dist_e(u,v)}{dist_g(u,v)}, \frac{dist_g(u,v)}{dist_e(u,v)})$$

When dealing with Small-World-Graphs, the distortion has only limited weight. In Small-World-Networks, clusters are supposed to exist, which are interlinked by so-called long-range-contacts. Although this contacts are far away in the sense of an embedding, they can be linked with high weight. The long-range contacts thus worsen the distortion as the term already suggests.

Allmusic-Styles To semantically measure the quality of an embedding, the allmusic-styles can again be used (see Section 4.4.2). Of course the semantic quality of the links have already been tested when constructing the graph, but to be sure that the embedding does not worsen this properties, such a measure can be built (see Algorithm 1). The distances of the styles and the distances in the embedding are constructed and their dependence is computed and summarized in a single value (called style_qm).

A value of 0.2 then means that for each increment of distance in the style-tree (see Figure 4.3), the distance in the embedding increases by 20 percent in average. The higher this average increase is, the better the songs of the “close-by” styles and the ones of the “far-away” styles get separated.

1: **for all** s_i, s_j **do**
2: // s_i and s_j have style-information
3: $dist_s \leftarrow level(lca(styleOf(s_i), styleOf(s_j)))$
4: $dist_e \leftarrow euclidean - distance(s_i, s_j)$
5: **end for**
6: $dist_t \leftarrow avg(dist_e \; where \; dist_s = t)$
7: **return** $avg(\frac{dist_{t+1} - dist_t}{dist_t})$

Algorithm 1: Quality measurement using style-information (style_qm)

5.2.4 Results

As shown in Table 5.2, it is reasonable to apply LMDS to the graph of songs, as the style-measure (which measures how well the songs of the same style get together and the ones from different styles get separated) is good. The songs from the same style are significantly nearer than the songs of completley different styles.

dim	landmarks	choosing	nodes	edge-distortion	style_qm
10	500	random	374738	1.704	0.210
10	500	max-min	374739	1.702	0.210

Table 5.2: Example embeddings with Landmark MDS

5.3 Iterative LMDS

A real-world graph which is created from imperfect data is of course itself imperfect. Some edges get into the graph which should not exist (e. g. between two very popular songs in a song-graph). The way the graph of songs is created in this thesis (see Chapter 4) lets strongly assume that some of those edges also exist in this graph. To locate these "bad" edges and remove them from the graph, we propose Iterative Embedding as a heuristc to take advantage of this fact and further improve the LMDS-embedding (see Algorithm 2).

5.3.1 Algorithm

After each embedding, the "worst" edges are removed from the graph. This process is repeated i times. After each iteration, a quality-measure has to be applied to the embedding to see if it is still improving or it already gets worse.

To locate the edges to remove, the stresses of all edges are computed. The edges with the largest stress are removed before computing the next embedding (see Algorithm 3). The fraction should be chosen as small as possible. The smaller the fraction gets the more iterations are needed to get to the best solution, so the lower bound for the fraction is set by the computation time.

Complexity: $O(n * l * d * i + l^3 * i)$, where i is the number of iterations.

```
1: landmarks ← createLandmarks(number, strategy)
2: coordinates ← LMDS(d, l)
3: for i = 0 to i do
4:    removeEdges(f)
5:    landmarks ← createLandmarks(l, strategy)
6:    coordinates ← LMDS(d, l)
7: end for
8: return coordinates
```

Algorithm 2: IterativeLMDS(dimensions d, landmarks l, fraction f, iterations i)

```
1: for all edges e do
2:    stress = length_embedding(e)/length_graph(e)
3: end for
4: for i = 0 to m * f do
5:    remove edge with maximal stress
6: end for
```

Algorithm 3: removeEdges(fraction f)

5.3.2 Experimental Results

When applying IterativeLMDS to the graph of songs, a significant improvement in the quality of the embedding can be observed.

In Figure 5.4 IterativeLMDS was applied to the full graph (about 370'000 nodes). In each iteration, 0.5 percent of the edges were removed. It is clearly visible that the quality of the embedding is improving. Therefore, different samples were made with different seeds for the pseudo-random-generator of the landmark-choosing-process (the graphs in Figure 5.3 show the mean quality over all samples). After about 30 iterations, the quality starts to decrease, since more and more "good" edges start to get removed.

The number of iterations to get to the peak seems to be independent of the size of the graph, when using a constant fraction of edges to remove at each iteration. Thus the fraction of "bad" edges stays constant as the graph grows. The evaluations with graphs of different sizes underline this assumption (see Figure 5.3).

To compare IterativeLMDS with LMDS, LMDS must be performed with $i * l$ landmarks, such that it has the same complexity. This is done in the example with 50 landmarks on the graph with 50000 nodes (see Figure 5.3). The peak is at about 30 iterations, thus $i * l$ is 1500. It shows that LMDS doesn't reach the quality that IterativeLMDS gives with the same complexity.

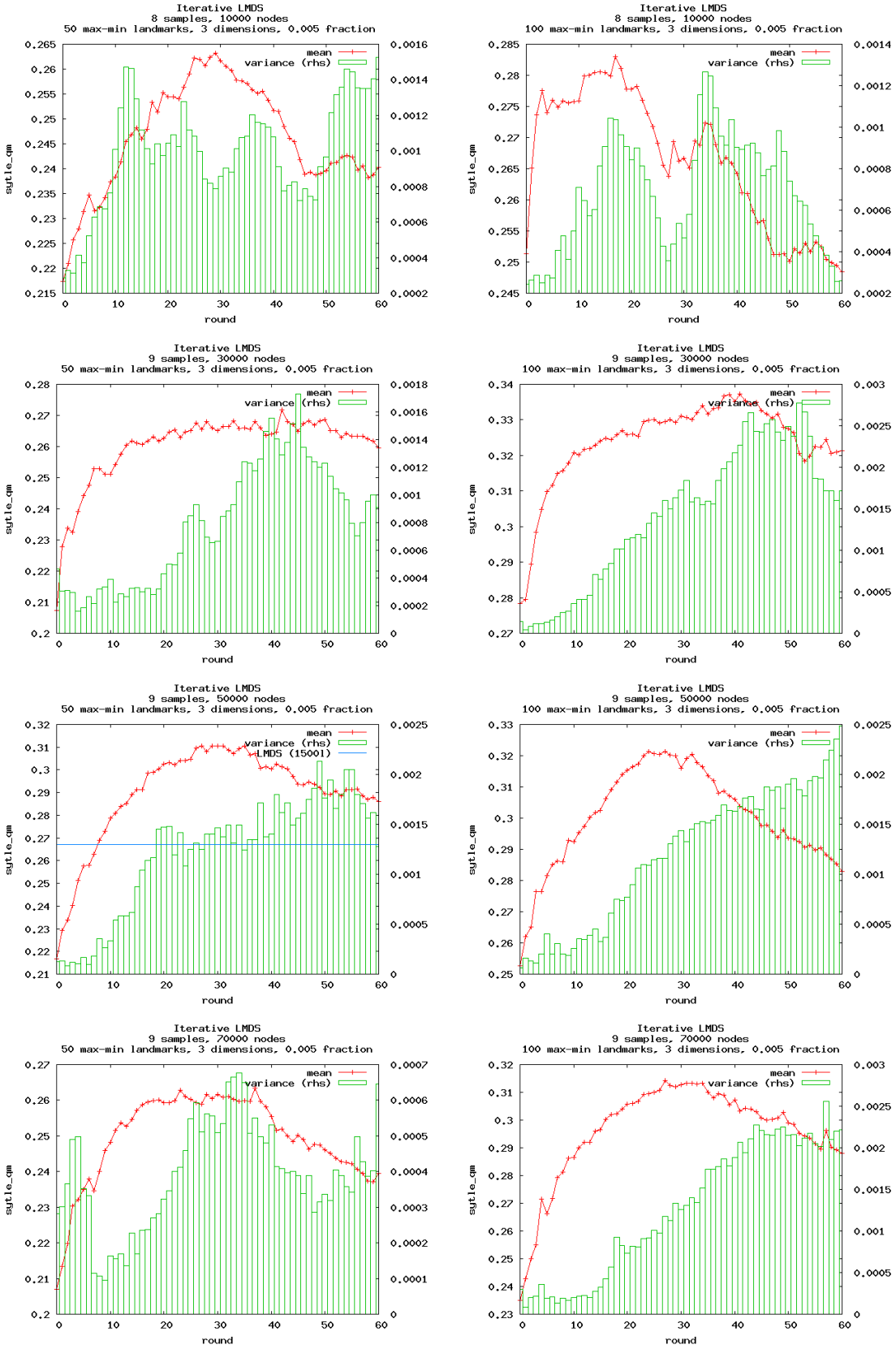

Figure 5.3: Simalutions of IterativeLMDS with different sizes of graphs. The bars show the variance over all the samples. The simulations show a clear increase of the quality (see Section 5.2.3) until a “peak” is reached.

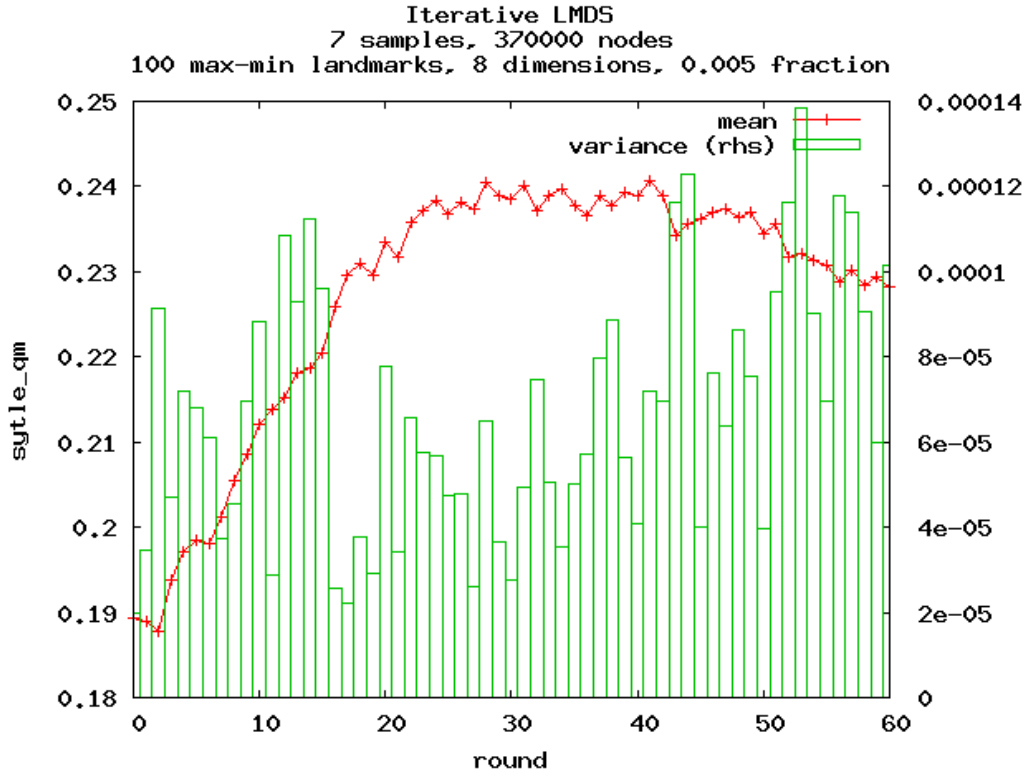

Figure 5.4: Simalution of IterativeLMDS with the full graph of songs (370000 nodes).

6 Application

Once an embedding of songs is at hand, some interesting applications become possible. The scenario is described in Figure 5.1. A user downloads the coordinates of his song-files and any application can use them in order to create a benefit for the user.

- Visualization of a user's music collection
- Recommendation Systems
- Playlist generation out of a user's music collection

A playlist-generator (see Section 6.1 and a style-proposer (see Section 6.2) were implemented and deployed in a web-application which is available at *http://pc-5413.ethz.ch:8180/musicweb* (see Section 6.3). In order to leave the embedding open for third-party developers, a web-service is available (see Section 6.3.1). Using the web-service, the coordinates for any song can be retrieved and processed to create new applications.

6.1 Playlist-Generator

The simplest algorithm one would think of for creating a playlist is described in 4. The virtual connection line between the start- and the end-song is divided into n pieces, where n is the desired length of the playlist. At each virtual intermediate node on this connection line, the nearest node is chosen. The nearest-neighbor-search is performed on a kd-tree[15].

This algorithms behaves badly if the start- and end-song are near and there are not enough songs in between them. In this case, songs around these two are taken in more or less arbitrary order. If the start- and end-song are even the same, the playlist contains the nearest neighbors of this song. In order to create a meaningful ordering, some kind of tour has to be taken.

compute connection line between $coords(v1)$ and $coords(v2)$
$intermediatePoints[] \leftarrow$ divide connection line into s-2 parts
$playlist[0] \leftarrow v1$
$playlist[s-1] \leftarrow v2$
for $i = 1$ to $s - 2$ **do**
    $n \leftarrow nearest(intermediatePoints[i])$
    while $playlist\ contains(n)$ **do**
        $n \leftarrow nextnearest(intermediatePoints[i])$
    end while
    $playlist[i] \leftarrow n$
end for
return $playlist$

Algorithm 4: Create playlist of size s from v1 to v2 from a given embedding coords

Artist	Title
The Beatles	All You Need Is Love
The Beatles	Lady Madonna
John Lennon	Intuition
The Beatles	In My Life
R.E.M.	Find the River
Gerry Rafferty	Right Down the Line
Neil Diamond	I'm a Believer
Frank Sinatra	New York, New York
Billy Joel	New York State of Mind
Stevie Nicks	Leather and Lace
David Bowie	Nature Boy
Alanis Morissette	21 Things I Want in a Lover
Mariah Carey	All I Want for Christmas
Fatboy Slim	Because We Can
Shakira	Hips Don't Lie (featuring Wyclef Jean)
Christina Aguilera	Make Over
Britney Spears	Boys
Britney Spears	Walk on By
Britney Spears	Sometimes

Table 6.1: Example playlist generated with Algorithm 4. Start- and endsong are "All you need is love" from "The Bealtes" and "Sometimes" from "Britney Spears".

6.1.1 Time-constraints

In some cases, the desired criterion for a playlist is not the number of songs in it, but the total length in minutes. E. g. if you want to play 30 minutes of music to fall asleep, or if you host a party and you want to play music for the next hour. For this purpose, Algorithm 4 has to be adapted.

1: $remainingTime \leftarrow d - duration(v1) - duration(v2)$
2: $playlist[0] \leftarrow v1$
3: $counter \leftarrow 0$
4: **while** $remainingTime > avgSongLength/2$ **do**
5: $guessedSize \leftarrow round(remainingTime - avgSongLength)$
6: $intermediatePoints[] \leftarrow$ divide $connectionLine(playlist[counter - 1], v2)$ into $guessedSize + 1$ parts
7: $n \leftarrow nearest(intermediatePoints[0])$
8: **while** $playlist\ contains(n)$ **do**
9: $n \leftarrow nextnearest(intermediatePoints[0])$
10: **end while**
11: $playlist[counter] \leftarrow n$
12: $remainingTime \leftarrow remainingTime - duration(n)$
13: **end while**
14: $playlist[counter + 1] \leftarrow v2$
15: **return** $playlist$

Algorithm 5: Create playlist of duration d from v1 to v2 from a given embedding coords

Algorithm 5 shows a possible generation of a playlist of a given duration. The size (number of songs) is guessed each time a song is added to the playlist. The remaining space between the current song and the endsong is divided by the guessed size and the next song is added. The duration of the generated playlist will not differ much from the given duration. In a bad case the duration of the very last song is a multiple of the average song length. As a consequence, the playlist will get too long.

6.2 Style-Proposer

For a small subset of the songs, the style is known (see Section 4.4.2). Using this knowledge, it is possible to make proposals for the other ones. There exist different methods to do this proposals:

Minimum average distance (avg) Propose the style whose songs have the minimum average distance to the input song.

Minimum average distance with x nearest songs (min) Take for each style the x nearest songs to the input songs and compute the average distance of these. Propose the style with the minimum average distance.

x **nearest neighbors (maj)** Take the x nearest neighbors of the input song and propose the style which the majority of neighbors has.

method	E[styleDistance] (see Section 4.4.2)
avg	1.877
min(1)	1.705985915492958
min(2)	1.601672535211268
min(3)	1.579665492957746
min(4)	1.587147887323944
min(5)	1.601452464788732
maj(100)	1.609222724031376
maj(150)	1.62621359223301
maj(200)	1.629384965831435

Table 6.2: Expected value of the style-distance between the proposed style and the real style of the songs for which style-information is available. Taking the three nearest songs for each style give the best results.

6.3 Web-Application

To make the algorithms and the embedding available, we developed a simple web-application. We therefore used some REST-Webservices[1] which are accessed with asynchronous calls[2]. The webservices then query the db-server that contain the metadata and the coordinates (see Figure 6.1).

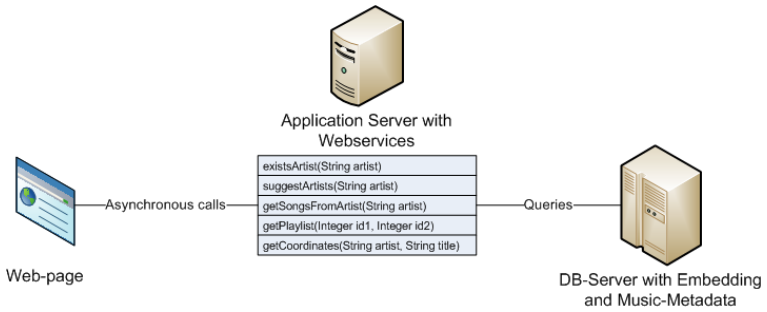

Figure 6.1: Architecture of the web-application.

With the web-application playlists can be generated from a startsong to an endsong. Optionally an intermediate-song can be chosen. The playlist is then generated from the startsong to the endsong via the intermediate-song. The length of the playlist can be stated either by a

[1] http://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm
[2] AJAX, asynchronous webservice-calls with Javascript

number of songs or by the length in minutes. Some sample-screenshots are in Figure 6.2. In addition to generate playlists, an option can be switched on to propose styles to the songs. This proposals are computed according to the algorithms described in Section 6.2.

The personal music-library can be uploaded via file-upload. For that reason, a file-format had to be created, since there exists nothing likely by now. An Xml-format that holds all the necessary meta-information of the songs seemed to be the best solution (see Section B). Once the library is uploaded, a session is created to store the personal information. Now playlists can be generated from a start- and end-song and afterwards downloaded in form of a .m3u-file[3], which can be played by most music-players.

The Xml-file has to be constructed out of the ID3-Tags[4] of the audio-files. This tags are often not complete or contain spelling errors. This problem can be faced with automated taggers, which synchronize the tags witch metadata-databases (e. g. musicbrainz.org) through acoustic fingerprints.

6.3.1 Web-Service

In order to allow a simple development of applications using the coordinates from the embedding of songs, a webservice is installed in the web-application. Thus any application can get the coordinates for songs and process them in a way that creates an additional benefit for the user. The URL-Schema is:

http://pc-5413.ethz.ch/musicweb/getCoordinates?artist=x&title=y

where x is the name of the artist, and y the title of the song (e. g. getCoordinates?artist=The Beatles&title=Yesterday) the http-response is in xml-format, containing a status-message and the coordinates.

[3] M3u is a file-format for playlists, originally created for Winamp.
[4] Meta-information attached to Mp3-files

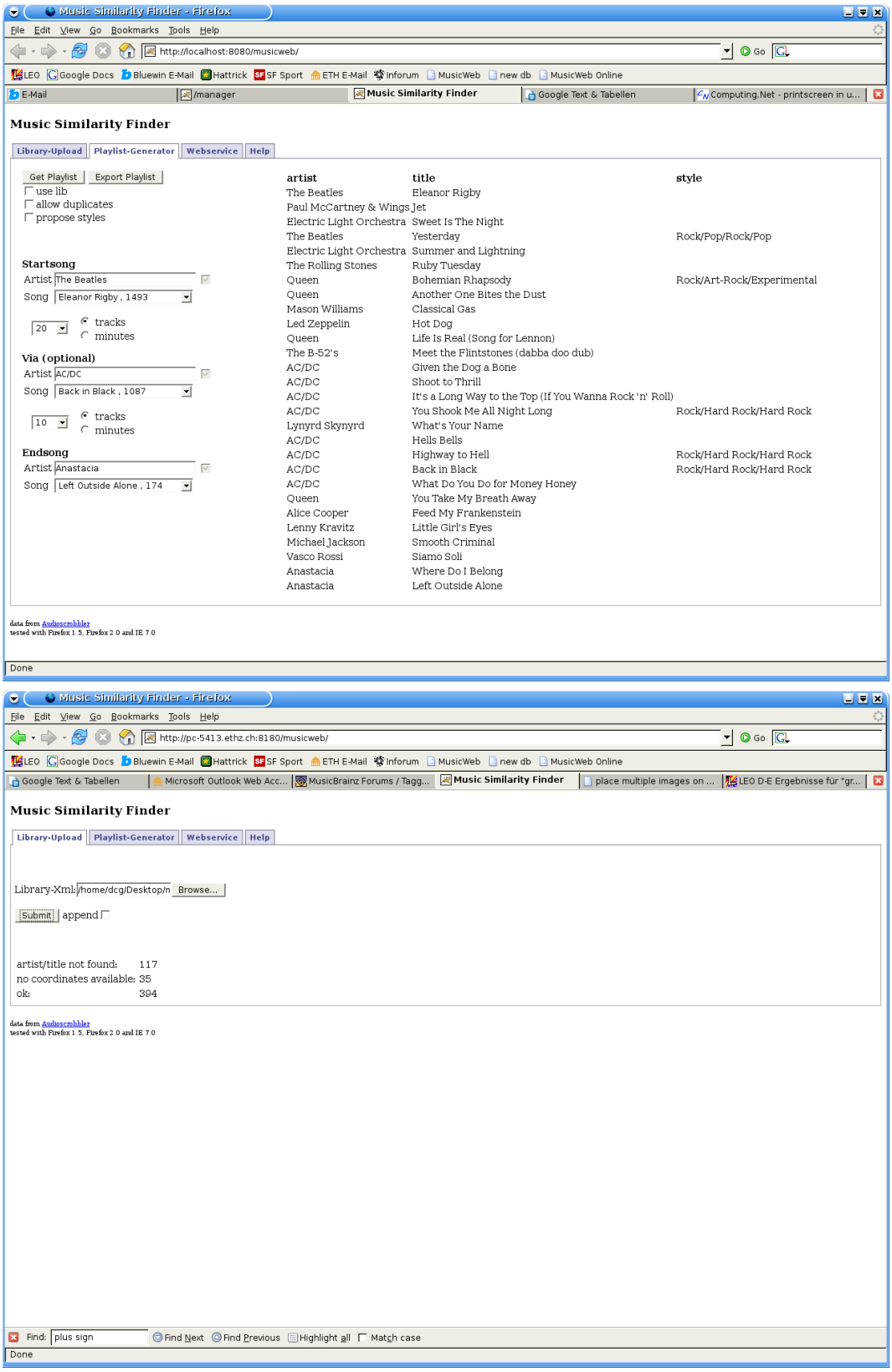

Figure 6.2: Screenshot of the web-application while generating playlists, and the upload of the library in the xml-format.

7 Summary and conclusions

7.1 Summary

As the quality measurements and the application show, it is possible to construct song-similarity and a graph of songs out of people's favorite (or most-heard) songs. The links created agree mostly with the assigned styles by the Allmusic.com and are far from being random. But to construct reasonable edges, a lot of data is required.

To get a notion of distance between two random songs, a distance labeling is most appropriate. Algorithms running on the graph itself are inpractical, since computing a single shortest path already has complexity $O(m + n * \log n))$ (n nodes, m edges). Out of the distance labeling and embedding algorithms, LMDS seemed to be a reasonable choice, because its complexity $O(n * l * d + l^3)$ is adjustable through the parameters l (number of landmarks) and d (number of dimensions). The quality measures showed that the embedding clusters "close-style"songs and separates "far-away-style"-songs.

The quality of the embedding can be further improved by applying IterativeLMDS, which iteratively embeds the graph and removed the "worst" edges (the ones with the highest stress). Having such an embedding, a lot of interesting applications become possible. In a demo-application, a useful playlist-generator was developed.

7.2 Future Work

More sources The graph is constructed in a incremental way (see Section 4.5). This "open" architecture allows to easily append new sources to the graph, other than last.fm. The occurrence-counter of the edge has just to be incremented each time two songs are stated as similar.

Incorrect/misspelled data The data provided by last.fm is often incorrect (see Table 3.2).

The misspelled names could be matched to correct ones using special alignment algorithms. Then less data would have to be omitted in the cleaning process and more would be available for constructing the graph.

Updatable embedding Since there steadily come in new nodes to the graph of songs, there is a need to insert this nodes to the embedding without constructing the whole embedding from scratch.

Community If there can be created a community (or use existing communities like musicbrainz.org), the graph could be made self-improving. Community members could increase or decrease the similarity of two songs.

Develop offline-application The web-application still is on a central server, but it would also be possible to download the coordinates of the songs once and store them. To ease the download a web-service was implemented (see Section 6.3.1). The coordinates could even be stored in custom ID3-Tags, such that they would propagate together with file-sharing of mp3-files. Once the coordinates are downloaded, the connection to the web-server or the db-server could be closed (gekappt), and the processing (playlist-generation, browsing, etc.) could happen local. Instead of developing an application a plug-in for an existing music-player (Winamp, iTunes) could reach a larger audience.

Compare with audio-based similarity measures The generated playlists and the similarities computed could be compared and evaluated with audio-based similarity measures. That would probably lead to new insights about the differences of the different approaches for music similarity (metadata-based, user-generated-data, audio-based).

Interest regions "Interest regions" of a user can be defined (spheres, ellipsoids, etc.), which reflect the regions in space a user is interested in. Music can be recommended that lies within these interest regions. In a peer-to-peer-environment, these interest regions can be intersect to retrieve the regions they both like to hear.

File-sharing Using the interest regions or another form of recommendation, a file-sharing application could be developed which recommends the files to share. Alternatively, the recommendations could be integrated into an existing file-sharing application as plug-in.

A
Development Environment

IDE

Java version: 1.5
IDE: Eclipse 3.2.1
DBMS: phpMyAdmin 2.9.0.2-Debian-1
JDBC version: 2.0

Database

MySQL server version: 5.0.24a-Debian_9-log
Storage engine: MyISAM

Application Server

Application server: Tomcat 5.5
Path: http://pc-5413.ethz.ch:8180
Web-Application: http://pc-5413.ethz.ch:8180/musicweb

Main-Classes

Crawling last.fm: *ch.ethz.dcg.ma.main.Crawler nr_users_to_expand nr_toptracks_to_fetch*

Processing users: *ch.ethz.dcg.ma.constructor.Constructor nr_users_to_process*

LMDS: *musicgraph.algorithms.MusicGraphLMDS host db dim nr_landmarks outfile*

weight_threshold occ_threshold seed

IterativeLMDS: *musicgraph.algorithms.MusicGraphIterativeLMDS dim nr_landmarks outfile weight_threshold occ_threshold seed filter_filename fraction iterations results_file sample_id*

B

Xml-Schema for the Library-Upload to the Web-Application

Xml-Schema of the file-format which is used by the web-application. An xml-file of this format can be uploaded and processed by the web-application. The “artist” and “title” are used to find the song in the database, “path” is used to create the .m3u-file that can be downloaded and processed by most music-players.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ethz.ch/dcg/musicworld/library"
xmlns="http://ethz.ch/dcg/musicworld/library">

<xsd:element name="songlibrary">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="song" type="song" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:complexType name="song">
<xsd:attribute name="type" type="filetypes"/>
<xsd:attribute name="artist" type="xsd:string"/>
<xsd:attribute name="title" type="xsd:string"/>
<xsd:attribute name="path" type="xsd:string"/>
<xsd:attribute name="length" type="xsd:integer"/>
</xsd:complexType>
```

```
<xsd:simpleType name="filetypes">
<xsd:restriction base="xsd:string">
   <xsd:enumeration value="mp3"/>
   <!-- add other formats ... -->
</xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

Bibliography

[1] J. Aucouturier and F. Pachet. Scaling up music playlist generation, 2002.

[2] J.-J. Aucouturier and F Pachet. Finding songs that sound the same. In *Proceedings of IEEE Benelux Workshop on Model based Processing and Coding of Audio*. University of Leuven, Belgium, November 2002. Invited Talk.

[3] Ricardo Baeza-Yates and Carlos Castillo. Link analysis in national Web domains. In Michel Beigbeder and Wai G. Yee, editors, *Workshop on Open Source Web Information Retrieval (OSWIR)*, pages 15–18, Compiegne, France, September 2005.

[4] Adam Berenzweig, Beth Logan, Daniel P. W. Ellis, and Brian P. W. Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Comput. Music J.*, 28(2):63–76, 2004.

[5] Matt Rasmussen David Gleich, Leonid Zhukov and Kevin Lang. The world of music: Sdp layout of high dimensional data. 2005.

[6] Vin de Silva and Joshua B. Tenenbaum. Sparse multidimensional scaling using land-mark points. 2004.

[7] Gerhard Widmer Elias Pampalk, Tim Pohle. Dynamic playlist generation based on skipping behavior. In *Proc. of the ISMIR Intl. Conf. on Music Information Retrieval*, pages 634–637, 2005.

[8] Gerhard Widmer Elias Pampalk, Tim Pohle. Generating similarity-based playlists using traveling salesman algorithms. In *Proceedings of the 8th International Conference on Digital Audio Effects (DAFx 2005)*, 2005.

[9] Daniel P. W. Ellis, Brian Whitman, Adam Berenzweig, and Steve Lawrence. The quest for ground truth in musical artist similarity. In *ISMIR*, 2002.

[10] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[11] Pawel Gajer, Michael T. Goodrich, and Stephen G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. *Comput. Geom. Theory Appl.*, 29(1):3–18, 2004.

[12] Jon Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, New York, NY, USA, 2000. ACM Press.

[13] Beth Logan. Content-based playlist generation: Exploratory, 2002.

[14] Yutaka Matsuo, Junichiro Mori, Masahiro Hamasaki, Keisuke Ishida, Takuichi Nishimura, Hideaki Takeda, Koiti Hasida, and Mitsuru Ishizuka. Polyphonet: an advanced social network extraction system from the web. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 397–406, New York, NY, USA, 2006. ACM Press.

[15] Andrew Moore. A tutorial on kd-trees. Extract from PhD Thesis, 1991. Available from http://www.cs.cmu.edu/simawm/papers.html.

[16] E. Pampalk, S. Dixon, and G. Widmer. On the evaluation of perceptual similarity measures for music. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFx'03), London, UK*, September 2003.

[17] J.C. Platt. Fast embedding of sparse music similarity graphs. In *Proc. Advances in Neural Information Processing Systems*, volume 16, 2004.

[18] John Platt. Fastmap, metricmap, and landmark mds are all nystrom algorithms. pages 261–268. Society for Artificial Intelligence and Statistics, 2005. (Available electronically at http://www.gatsby.ucl.ac.uk/aistats/).

[19] R. Ragno, C. J. C. Burges, and C. Herley. Inferring similarity between music objects with application to playlist generation. In *MIR '05: Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 73–80, New York, NY, USA, 2005. ACM Press.

[20] Fabio Vignoli Rob van Gulik. Visual playlist generation on the artist map. In *Proc. of the ISMIR Intl. Conf. on Music Information Retrieval*, pages 520–523, 2005.

[21] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–+, June 1998.