Jukebox

An Intelligent Online Music Player

Semester Project,

March 2009 – April 2009

Student: Mihai Calin

ETH – ITET Master Studies

Advisors: Olga Goussevskaia / Michael Kuhn

Supervisor: Prof. Dr. Roger Wattenhofer

Task Formulation

Media usage is changing rapidly these days. This process has been ignited by several technological advances, in particular, the availability of broadband internet, the World Wide Web, affordable mass storage, and high-quality media formats, such as mp3. Many music lovers have now accumulated collections of music that have reached sizes that make it hard to maintain an overview of the data by just browsing hierarchies of folders and searching by song title or album. Search methods based on song similarity offer an alternative, allowing users to abstract from manually assigned metadata, such as, frequently imprecise or incorrect, genre information. In a context where music collections grow and change rapidly, the similarity-based organization has also the advantage of providing easy navigation and retrieval of new items, even without knowing songs by name. This opens possibilities, such as sophisticated recommendations, context-aware retrieval, and discovery of new genres and tendencies.

In previous projects we have created a Euclidean map of the world of music, which contains more than 400K songs. This map places similar songs close to each other, whereas the distance between unrelated songs becomes large. Such a map exhibits several advantages in terms of applications. It allows to quickly find songs similar to each other, to define regions of interest, or to create smooth playlists by following paths, such as straight lines between start- and end-songs, for example. Based on this map, we have developed the music-explorer website (www.musicexplorer.org) that provides a similarity based view on music collections.

The current website, however, provides only limited benefits to the user. In particular, it is not able to directly play music. The goal of this thesis is to improve in this point by connecting the music-explorer website to existing web-content. In a first step, Mihai will study different APIs, such as the last.fm, youtube, google, or facebook API, and investigate how the music-explorer project could benefit from each of these data sources.

Dependent on the outcome of the API evaluation, Mihai will then design one or several new use-cases for the music-explorer website that involve some of the evaluated data sources. In a last step, Mihai will then implement at least one of the proposed use cases and make it available to the user on www.musicexplorer.org.

Abstract

The continuous growing of people's music library requires more advanced ways of computing playlists through algorithms that match tracks to the user's preferences. Several approaches have been made to enhance the user's listening experience; while most of them rely on the music content provided by the user, this project presents an online application that sources the audio content from publicly available resources (YouTube). A playlist generation algorithm is developed that uses only one seed track to compute a playlist of arbitrary length. For sourcing the audio content, YouTube's track coverage is analyzed and statistics show that, in a real-life usage scenario, almost 80% of the tracks are available while the rest have rather lower popularity. The resulting application is a fully functional but feature limited online music player that can also serve as a framework for future playlist generating algorithms or other content sources.

Overview

1 Introduction

Introducing the current report and placing it aside of similar ETH internal and other works. Some introductory words about the music space, the goal and the approach used in this report.

2 Problem Statement

This chapter debates the necessity of a new solution to make the world of intelligent track comparison even more accessible to the end user, followed by an approach sketch.

3 Related Work

When a new, revolutionary, product comes to market it usually has a mixed impact on the audience. This chapter analyzes some of the achievements but also flaws of existing solutions and therefore represents a base for the discussion in the Vision Chapter.

4 Vision

This chapter sets the ground rules for developing the application, mostly based on experience gathered from already existing implementations. Important decisions about audio content sourcing and application promotion are discussed and reasoned in this and also the next chapter.

5 Audio and Video Content Provider

This chapter analyzes advantages and disadvantages of the different content providers and presents a coverage analysis for YouTube.

6 Architecture

After discussing the application's approach to serving its goal, it's time to describe the technical details of the implementation. What functions should the client and server side implement? How modularized can and should the application be build? What's the best approach in dealing with the huge data amount? What advantages do the different APIs bring and how can they be used?

7 Conclusions

The conclusions section summarizes the projects final state as well as reconsiders the main ideas and results.

8 Discussion and Future Work

The Section deals with the ideas that didn't make it to the development process mainly because of the tight time scheduler, but that are important to mention and at some point or another in a future development they have to be addressed.

Table of Contents

1 Introduction

Music has always been a means of entertaining people even from the earliest ages of the civilization. Historically it was produced by musicians and only available during life concerts. The technological evolution made it possible to save the music on vinyl plates, later electromagnetic charged stripes, CDs until the technology brought us to saving tracks digitally. When dealing with a huge collection of tracks, people encounter management problems they did not have before. So they have to develop new ways of using the music collection for their entertainment. Playlists are a good approach for saving successions of tracks that one likes. The most dominant problem of existing playlist generation mechanisms is, however, their lack of flexibility: new tracks are not automatically added, they don't adapt to the user's current mood etc.

A new approach in dynamically organizing tracks into playlists is on its way: companies like last.fm already suggest an algorithm of mapping songs one to each other based on their "similarities"; but how to compute these similarities? One way, that did not prove to be very productive, is to analyze the audio content of the track – its audio frequencies. This way, tracks are split in categories like "Heavy Metal" and "Blues", but people do not like all tracks of a certain gender and these genders might be inaccurate. Another way, which is given more and more attention by researchers and companies worldwide, is computing similarities between tracks based on user input. As an example: if two users add the same two tracks to their playlists, one can deduce that these tracks are similar and so, also other people that pick one of them are likely to enjoy the other one as well.

Lorenzi (Lorenzi, 2007) proposes a way of representing the similarity between tracks in a 10-dimensional Euclidian space (further called music space), where the closeness of tracks is approximately proportional to their similarity. 7M songs currently appear in the database, but only 500K of them have enough user statistics to be mapped in the graph. Using this simplified and computationally efficient way of finding similar tracks, several applications can explore new ways of computing playlists. Most of them offer support in playlist generation but none also provides the tracks to be played. This could be seen as a disadvantage because not all people possess all tracks that are suggested by the space.

This report sets its goal in developing an application that uses the space of music for computing intelligent playlists and, more important, trying to deliver the required songs to the user through publicly available tracks on YouTube.com. For promotion purposes and also user data gathering, it was chosen to make the application available on Facebook[1], which ensures an easy referral to friends.

The development process is not an easy one as many technologies are required to work together to provide the user with a good experience. The huge amount of data that has to be processed is demanding with respect to both algorithms and

[1] www.facebook.com - community networking site

underlying hardware. In the end of this report, a fully functional, but feature-limited, version of an online music player is presented. This application could be the step stone of future, more detail-oriented applications because it sets the basis of a new playground: an online music player.

2 Problem Statement

This chapter debates the necessity of a new solution to make the world of intelligent track comparison even more accessible to the end user, followed by an approach sketch.

2.1 Motivation

Several solutions already use intelligent playlists embedded in music players installed on computers. There are also online solutions, the most popular of which is last.fm, which acts as a personalized radio station that plays preferred music. On the other hand it does not allow playback of a certain track. There are also other solutions, like the genius function of iTunes or the Music Explorer; both use the user's music collection to generate playlists. The biggest disadvantage of the latter solution is that the user can use only tracks that he/she already has on his/her PC to generate playlists. Of course this limits the power or the algorithm very much.

There are already services that provide the music content (like last.fm or YouTube to name a few) so it's a natural conclusion to try to use these services in connection with the playlist-generating algorithm.

In order to understand the utility of such an application, just imagine the following scenario: one enjoys listening to music while working. It is not common to store music on the company's computer so one rather has a personal mp3 player with himself during office time. If one takes enough time to prepare ones playlists in order to fit ones current mood, it is a pretty decent solution. But what if new tracks appear that one might like? One first has to do serious research in order to find them and then go through buying them, downloading them to his/her mp3 player, updating the playlists… it already sounds very difficult, right? Now the suggested scenario is the following: one opens a web site, types in a track that reflects ones current mood and hits "play". That's it! The player chooses tracks that one likes, also plays new tracks that one did not hear before, and can go like this for hours and hours without repetition. One can go on with one's work and in order to stop the music, one only has to hit stop or close the browser. The simplicity of the solution speaks for itself.

The goal of this thesis is to analyze and implement an approach of building such a web-based music player. The questions it has to find answers for are: How should the user interaction be designed to maximize the user satisfaction? Where to source the audio (and video) data from, while ensuring a maximum coverage? And finally, how to promote the application in order to attract as many users as possible?

The different implementation possibilities are evaluated and the best solution is implemented. The logic behind the web-based music player computes a

sequence of tracks based on their similarity. At the same time, user behavioral data is gathered that helps further releases to be even more user friendly. Another important aspect of the application is its extensibility. Modularity and code reusing are very important parameters of this application, as it acts as a version 1.0 for future releases. These future releases will be able to interact with the user for finding the best track video on YouTube or to determine the music preferences of users and even adapt the space to the new usage statistics.

2.2 Approach

The analysis of the currently available tools to accomplish the task is one of the most important steps because the ground concepts of the application should never change, regardless of its future complexity.

The several possible implementations of the web service together with the balancing of computing tasks between server and client are the first parameters that have to be defined for a solid base. Also the programming language plays a crucial part in the development process, as it is shown later. The amount of callbacks to the database in favor of less memory usage is also an important aspect that is difficult to estimate from the start. In order to allow a high flexibility while still maintaining a small dataflow, the implementation of the logic is mainly on the server. The UI responsibility is fully retained by the client side as well as servicing UI requests and only notify the server of such activity.

In order to achieve the high goals that were set, the structure of the application is important to be highly modularized to allow interchanging the modules with better, more complex implementations. It is important to determine which components are possible and also easy to modularize, without introducing too much communication overhead in the interfaces. It turns out that the music content related jobs can easily be modularized, as well as the DB related jobs and the playlist computing tasks. The core of the application only needs to handle these modules and the logging task. Also the communication with the client is modularized, making it particularly easy to implement new clients running on the same service or new services to serve the same client.

One of the hardest tasks is determining how to provide the audio content to the user. The playlist computing is based on a music space implementation, described later. For the audio content there is anyway no guarantee because it is an externalized task. Several providers of audio content are analyzed and the best one implemented in such a way that it is reliable for the users.

For the client side, JavaScript and Flash are the only two solutions to provide a good user experience because there is no need to install any software. When working with JavaScript, AJAX (Wikipedia, AJAX (programming), 2008) is needed for the seamless communication between client and server – thus enabling a richer user experience – and also for avoiding communication overhead. GWT (Google, 2008) dramatically eases the interaction and object exchange between the client side and the web service because this tool enables the programming of the client side in java and converting it to JavaScript.

Finally, one of the most important steps in the future extension of this application is the logging of user activity. The activity of the users has to be

logged in such a manner that constructive information can be extracted from the logs. It's important to build the logs right such that future ideas can find the data to base their research on. For this step no resource is too expensive. One has to differentiate between user activity and user feedback. User activity can be represented through actions like: jumping to a certain track, rearranging the generated playlist; user feedback is the action of notifying the application of a bad chosen track or of a track that the user doesn't like. In the end the logs are saved in the database and their exact structure discussed.

In the end of the report, an implementation is presented showing a web based, personalized, music player. It is an interactive web site where the user selects a favorite track and generates a playlist based on this. The playlist has the following properties:

- Each track in the playlist is similar to the one before;
- A track is only allowed to be repeated if it has not been in the last 24 tracks; this should guarantee about two hours of non-repeating tracks;
- Tracks from the same artist do not appear more than at least 4 tracks apart;
- The user has the possibility to reorder the tracks, to remove them or to play them directly, without waiting for their turn to come;
- The music video of the currently playing track appears on the right;
- The user has the possibility to open the playing track in YouTube (with possibility to switch to full screen).
- The user has the possibility to give feedback about the quality of the recording or weather he/she likes the track or not;
- All user activity like reordering playlist, searching for tracks, track removal – but also his/her direct feedback – is logged for future analysis.

After describing some of the most important features the application has to cover, an analysis of already existing solutions is made in the Related Work Chapter.

3 Related Work

When a new, revolutionary, product comes to market it usually has a mixed impact on the audience. This chapter analyzes some of the achievements but also flaws of existing solutions and therefore represents a base for the discussion in the Vision Chapter.

3.1 ETH Projects

Several projects belonging to the Distributed Computing Group, part of the Electrical Engineering Institute of the ETH Zürich, have presented different approaches as a playlist generating tool based on a specially created music space.

The first and most important was the creation of the music space itself (Lorenzi, 2007). Based on user statistics provided by Last.fm, Lorenzi presented a way of representing the data such that the similarity between two tracks can be quantified. By mapping tracks to points in a 10-dimensinoal space, the similarity of two tracks can be easily computed as the Euclidian distance between them. Through this, computationally hard problems become easy accessible for other

applications, such as: comparing similarities between several track pairs, computing the nearest neighbor, finding tracks along a certain path or "negating" a track – meaning finding another track that is far away from this track. A further discussion of the correctness of this mapping is out of the scope of this report; therefore this report considers the provided music space as efficient and uses it as it is.

Based on the previously described music space, several applications were developed to bring the advantages of the music space to the end user. The musicexplorer web application (Gonukula, Kuederli, & Pasquier, 2008) presents an approach of organizing the user's music collection based on an online service. The user has the possibility to upload the titles of its music collection, select two of them and the algorithm computes a playlist that is a smooth crossover from one track to the other. Both, the musicexplorer and the application presented in this report, allow the user to generate intelligent playlists. The downside of the musicexplorer is the requirement that the user already possesses the tracks to be played. If his/her collection is limited, so is the playlists at the output. Using publicly available tracks not only make it easier for the user but also the results are be better.

Bossard presents a new way of exploring the Space of Music with a visual approach that should allow the user to visually select tracks from the music space (Bossard L. , 2008). The ideas are implemented on the android platform through a customizable music player.

3.2 External Projects

Several companies have recently discovered the advantages of user-personalized playlists and are currently building their business model based on providing high quality but simple to use solutions to their customers. Last.fm is one of the first ones on the market to offer personalized music playback; it achieves this through an online "radio station" that is supposed to play music that the user likes. The Last.fm approach is very similar to the one proposed by this report: it is an online music player that usually plays the user's favorite tracks and also introduce new tracks if available. However, because the personalization of the player is done on a per-user basis (opposed to the per-session proposal of this report) it could happen that it does not react so fast to mood changes in the user's behavior. The biggest drawback is still the inability of the user to select certain tracks to play; even if he/she has a certain favorite track in mind, he/she has to stick to the automatically generated playlist of the player. This limitation is probably more license-based than technological, but it is a field where improvement is desired.

Another approach to delivering similar tracks to a user is the Genius function of the Apple iTunes music player. The principle is fairly easy: iTunes analyzes your music library, submits the tracks to an iTunes server, which returns popularity measurements personalized for your tracks. During playback, the user has the possibility to select a track and let the application generate a playlist containing tracks similar to the selected one. The biggest drawback of this approach is the limitation to the user's personal computer, where he/she holds his/her private music collection.

During the development process, two new companies launched an implementation that is very similar to the presented approach. One of them is Songza (www.songza.com), an online music player where users can search for a track and play it. The audio (and sometimes also video) content is sourced from YouTube or Last.fm. The user has the possibility to manually create a playlist. However no mechanism of automatically generating playlists is yet available. The second, and much more similar to this report's implementation, is DropPlay (www.dropplay.com). It allows the user to search for songs, generate playlists based on similarity and also share these songs with friends on Facebook. Only time will show which of the two implementations will be better for the users.

The discussion of external projects is continued in the Vision chapter, where the pros and cons of the mentioned applications are further analyzed and weighted.

4 Vision

This chapter sets the ground rules for developing the application, mostly based on experience gathered from already existing implementations. Important decisions about audio content sourcing and application promotion are discussed and reasoned in this and also the next chapter.

4.1 Requirement Specifications

As presented in the related work section, there already are several implementations that are more or less similar to this report's solution. All of them have their strengths and weaknesses so they are going to be analyzed and combined to work in the targeted scenario. The goal is to create a web based application that is simple to use and that can function independent of the users resources. What can be learned from other approaches and what can be used to increase the value of this implementation?

Compared to the current musicexplorer web application, the designed application is also a web-based service and targets the same audience: people who want to have their music experience enhanced without a lot of effort on their side. The users interact with the existing musicexplorer application by submitting all the titles of their music library and then selecting two of those tracks that are used as endpoints of a playlist. All the tracks in between represent a smooth progress from one track to the other. This implementation faces the problem that it completely depends on the user to provide the audio content. The quality of the offered result is only as good as the number and variety of submitted tracks; even the best implementation performs poorly if it is fed with low quality resources. Also the mapping of tracks from the user's tags[2] to the applications tags is a problem to take into account. The lack of own music content is also a problem for people that do not own a big music collection or do not have it at hand. Hence, the current application sets a high priority to

[2] Track meta-data, namely artist and title

providing the intelligent playlists together with the audio content. This ensures both a greater QoS[3] and ease of use.

Generating an intelligent playlist and then playing the tracks is similar to Last.fm. In the case of Last.fm however, license limitations prevent the user to search for specific tracks and play them. As long as the designed application has no restriction in this matter, it is a high priority to implement such a popular feature. A search bar allows the user to browse through the library and select individual tracks to play. Last.fm also avoids displaying an actual playlist, although most users like to at least see the tracks that follow, and possibly even edit this list. To honor the playlist generating feature of the application, a playlist is displayed to the user together with the possibility to add, remove and rearrange tracks from the list. This makes the application very similar to the music players the user is already used to.

As discussed in the last paragraph, the UI[4] imitates a common music player such that the users do not have to get used to something completely new. This brings the question wheater to implement the application as an installable – standalone application or as a web service running in a browser. Modern technologies like JavaScript and AJAX allow a simple development of a solution running in the web browser. Also encouraging for using this approach is the fact that the application most likely provides online-based streamed audio content. Online applications provide decent results if they don't have resource-consuming client side tasks; also they are easy to update and maintain. In respect to all the mentioned pros, it makes sense to develop the application as a JavaScript based implementation that runs directly in the web browser, without the need to download or install anything.

4.2 User Perspective

As one of the important goals is to have a user-friendly application, it is necessary to provide good results with a minimum of information requested from the user. As a second, but also very important feature, the user actions have to be logged for understanding and improving the application's behavior. The logging should be made optional, to avoid privacy issues.

Understanding the resources that are at disposal is necessary to find the minimum information required to return satisfactory results. In order to provide tracks that the user likes to listen to, the application has to know which are the preferences of the user. Last.fm does that by tracking the user's preferred tracks over several sessions and probably computes a user profile based on this behavior. This approach is an elaborate one but introduces a huge overhead in the user-management algorithms. And if the user decides at some point to listen to something else, the inertia of the algorithm makes this very difficult. The best approach that also ensures a minimum user interaction is no user tracking at all. The application should work well for each user, independent on his/her history with the application.

---

[3] Quality of Service
[4] User interface

Choosing not to implement any user management requires obtaining as much information as possible at the start of each session. In order to make it very flexible, the application can ask for "guidelines" (favorite tracks of the user) each time it's requested to compute a new playlist. The approach of another ETH project was to ask the user for several seed[5] tracks and computing a virtual path in the music space that connected all the points. Of course the seed list was also rearranged such that the resulting path is minimal. Using several seeds is a good approach because it gives more details about the user's current music preference. But on the other side, the user might feel overwhelmed by the information he/she has to provide to create a simple playlist. In respect to that, the first approach is to ask the user for only one seed track. This track is used to create a new playlist, and if the user doesn't like the offered tracks he/she can insert a new seed or just reshuffle. Tests will show if this minimal user input approach is efficient enough in providing good tracks.

User feedback is very important for improving this application but it is well known that the average user does not usually bother to give feedback; and if he/she does, he/she usually reports features that are not working. To adapt to this attitude, it is good to implement a way of monitoring the user's actions, like when he/she changed the track, when he/she removed a track from the proposed ones, etc. Also the user has to be given the possibility to report faulty features. Because the music content provider is YouTube and no guarantee can be given for the played content, the user has the possibility to report tracks that don't fulfill his/her expectations. For example, if a track is a low quality, concert recorded version, the user can notify the application and other users do not receive it anymore. Another complaint possibility is about the playlist-generating algorithm itself. The user is able to report a track that he/she doesn't like so much, indicating an inconsistency in the playlist generating algorithm or the underlying music space.

4.3 Implementation as Facebook App

Promoting the application to many users is important – among other reasons also because user's behavior is recorded and eventually leads to a better application. Applications usually get promoted through commercials, referral programs etc. As the application is one of a kind, word of mouth is the best way to promote it. Community networks like Facebook and MySpace[6] are perfect environments for deploying and promoting such community-targeted applications. This networking platforms already have thousands of daily returning users signed up and also facilitate the integration of applications through specially designed APIs. Facebook APIs also offer the possibility to read user data; this could come in handy at a later development stage. Because the possibility of recommending applications between the users is provided by the Facebook platform, the application is basically promoting itself – provided it proves it's usefulness. Also appearing on a Facebook official site, the application

---

[5] Seeds or seed tracks refer to the track provided by the user and used by the playlist generating algorithm to find similar tracks.

[6] www.myspace.com - community networking site

is granted more trust from the users, making it possible to integrate it even better with the community.

Facebook offers two possibilities of designing an application: either the app is written in a Facebook-specific markup language or any other web site. If the latter is chosen, the application runs in an iFrame, of maximum 760 visible pixels in width. Because the application is intended to run also as a stand-alone web site, the deployment in an iFrame is chosen.

Choosing the promotion through Facebook doesn't limit the application in any way. It still runs in a web browser without any restrictions and independent of Facebook. One can see this deployment as an add-on to the functionality.

Also the DropPlay application, presented in the Related Works Chapter, integrates with Facebook and uses Facebook to recommend tracks to friends (and through this indirectly refer the site to the friends). However, the DropPlay application is not deployed as a Facebook app; not doing so neglects the trusty environment for the users. To speculate upon the reasons, the width of the iFrame might be too restrictive for DropPlay's UI.

Having discussed the ground rules of functionality and also reasoned the decisions through arguments mainly from the users perspective, the spotlight continues towards the sourcing of the audio content.

5 Audio and Video Content Provider

This chapter analyzes advantages and disadvantages of the different content providers and presents a coverage analysis for YouTube.

5.1 Content Provider Analysis

The necessity of providing public music content was analyzed earlier in this document, concluding that it is important for a good user experience. To be taken into consideration, a music provider has to fulfill the condition that it offers a complete (or high) coverage of the track database. With respect to this requirement, only YouTube and Last.fm can be considered good candidates at this moment.

YouTube's pros are: an easy to use API for implementing the video content in the web browser; a java library for making the search for tracks easy and fast; the tracks are uploaded by users, which means the data is always up to date with the latest tracks. On the downside, because the content is user-generated, the quality of some content is so bad, a results post-processing algorithm is necessary. Also not all tracks are found on YouTube, but popular tracks are usually available.

Last.fm has a better coverage of the tracks in the music space and the quality of the content is also very high. The biggest downside of Last.fm is that the majority of tracks are only 30 seconds snippets of the actual tracks. This is something unacceptable for a music player, thus turning the decision in favor of YouTube.

The developers of Songza couldn't apparently decide for one of the providers and implemented both. It is a good approach because the songs on Last.fm have a

guaranteed quality and the application only switches to YouTube if there is no unclipped track on Last.fm. Implementing both alternatives however, brings problems especially in the UI part of the application. Also indirect user feedback is much harder to trace. This is why the application is relying on YouTube as it's content provider.

5.2 YouTube API

5.2.1 Technical Possibilities

As discussed in the previous section, YouTube is the chosen music content provider. It should be noted at this point that no music content is being held or even transferred through the server hosting the service. Only the YouTube movie id is passed to the client application that embeds the video using YouTube's own player.

The YouTube API offers the possibility to do most of the actions performable manually through the web site in a programmatic manner. These actions include receiving a list of results to a search query and for each of these tracks: its id, the link to its implementation, link to its thumbnail etc. Moreover, the API offers methods to set the query parameters, like: the country restriction, the language, the uploader and also the order of the results.

The API also offers possibilities of logging in with a user account, uploading videos from that account, generating playlists and many others features, but they are (currently) of no use to this application.

5.2.2 Coverage Analysis

YouTube was chosen over Last.fm to be the sole music content provider of the project both because of its simple to use API but also for covering most of the popular tracks. Coverage is one of the strong points of Last.fm, but the fact that for most tracks only a 30 seconds sample is provided, biased the decision in YouTube's favor.

To analyze to what extend the library is covered by tracks from YouTube a test was run on samples taken from the DB. In order to have an accurate result, a certain number of random samples were chosen from the DB in order to make an assumption about the coverage. The results show that from 137 samples, more than half had no corresponding track in YouTube (Figure 5-1).

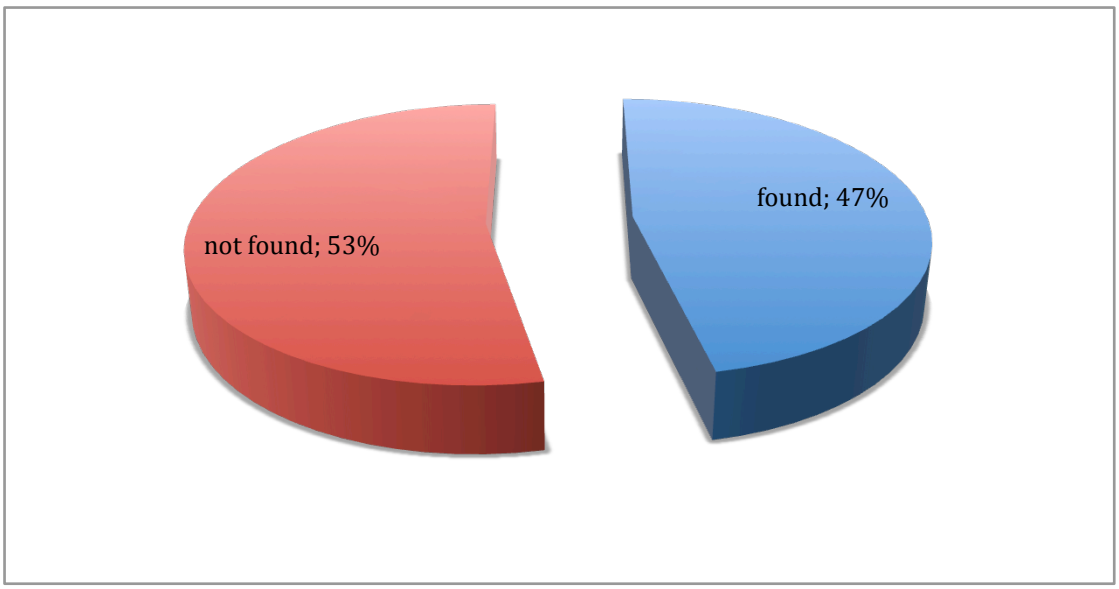

Figure 5-1 - YouTube Track Coverage

The coverage test was run over 7M+ tracks available in the DB, randomly choosing 137 tracks. But around 60% of the tracks have a popularity of 1, meaning that only one user ever listen to them. As a comparison, the average popularity of the rest of 40% of the tracks is 24 with a maximum of around 62K. The statistic doesn't necessary reflect the real coverage of the DB, because most users only listen to the most popular tracks – also enforced by the playlist generation algorithm that select mostly rather popular tracks. Hence, in order to reflect the effective coverage, a new statistic approach is chosen, detailed next.

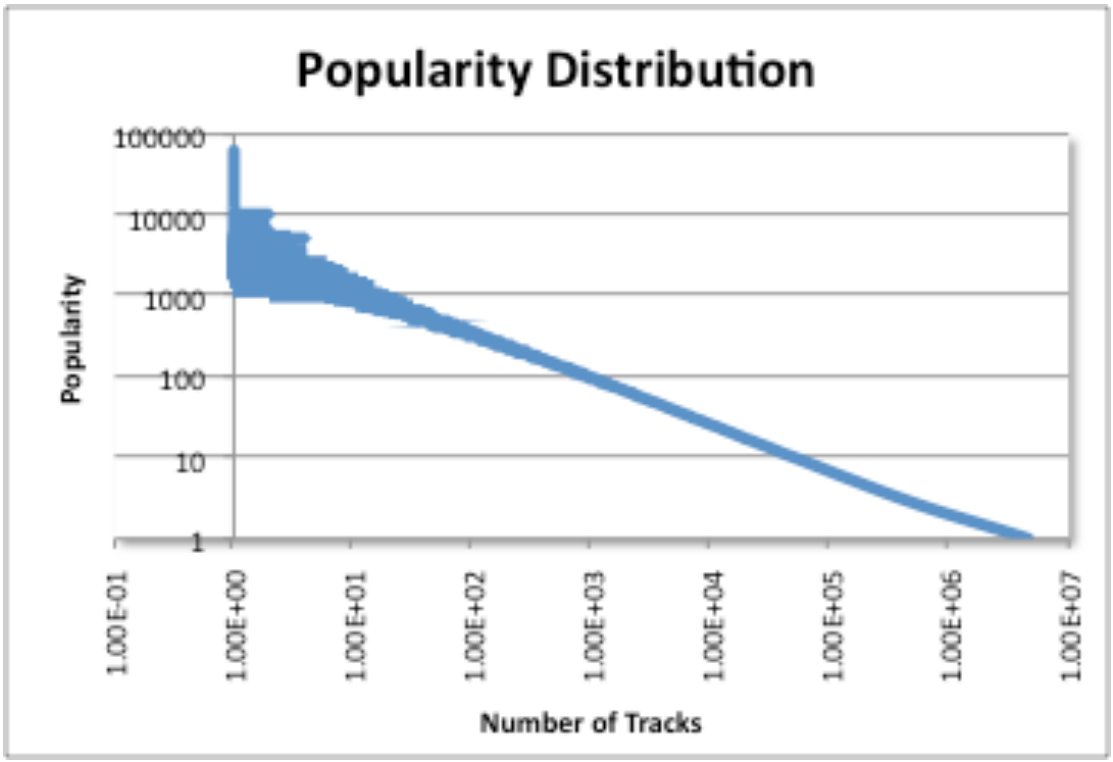

Figure 5-2 - DB Tracks Popularity Distribution

Figure 5-2 illustrates the popularity distribution in the database. From the plot one can for example see that more than 100K tracks have popularity above 10. In order to create a weighted analysis of the coverage, tracks for analysis were chosen in the following manner: the list of tracks was sorted by popularity and the first 10 tracks were added to the statistic. Next, 10 tracks were chosen randomly from the next power of 10 tracks (for the second step 100) and so on until the limit of 7M tracks was reached and the sample contained 191 tracks.

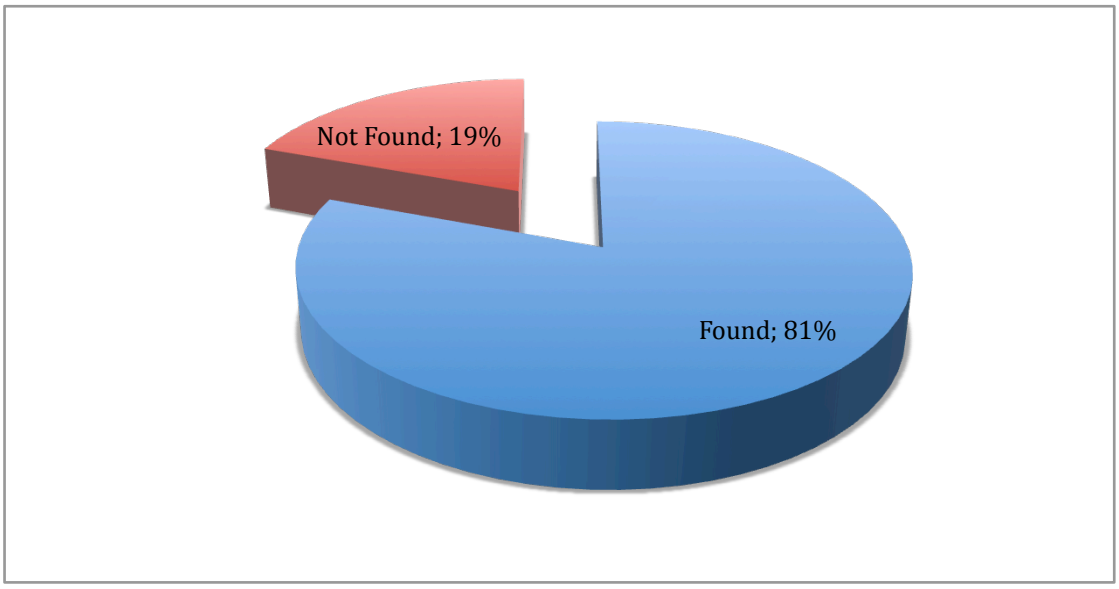

Figure 5-3 - YouTube Weighted Track Coverage

Figure 5-3 shows the YouTube coverage the weighted statistic discussed earlier. Only 19% of the tracks are not found at all on YouTube, and most of them are in a lower popularity range, as shown in Figure 5-4 - YouTube Popularity vs Coverage.

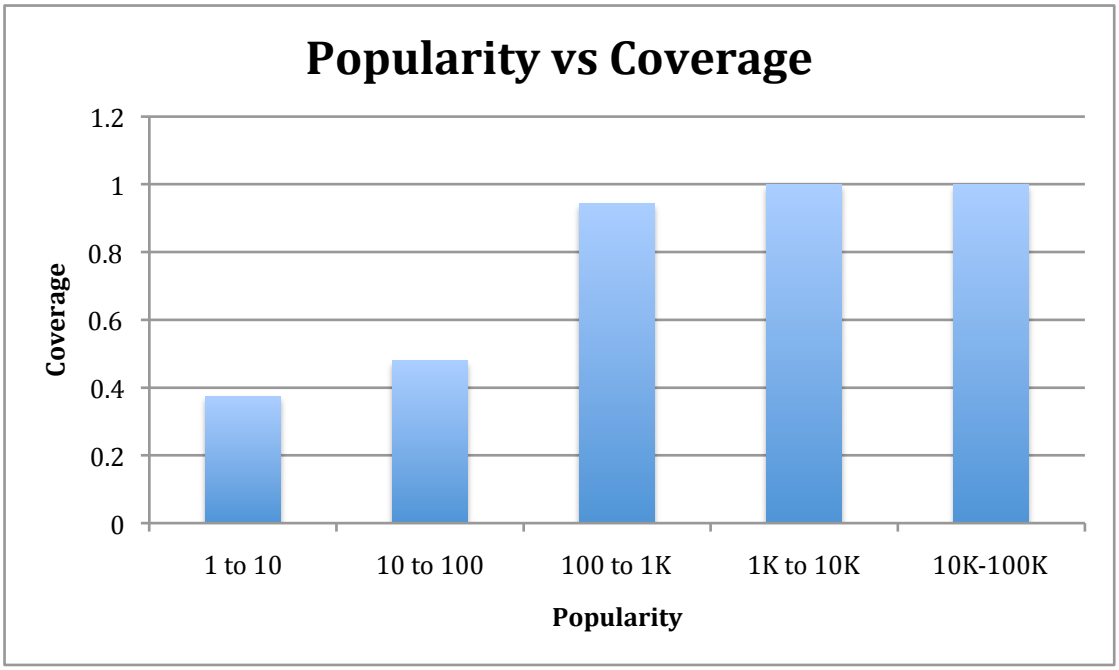

Figure 5-4 - YouTube Popularity vs Coverage

5.3 Query Design

As a first approach, the query sent to YouTube contains only the track's artist and title, concatenated in a string and separated by space. This was thought of as a first try but the results returned by YouTube are good enough to continue with this solution in the development environment.

Other parameters were also supplied to the query, like to return only videos that are visible in the client's country. To achieve this, the clients IP address is provided together with the query. Also the result's sorting is requested to be by 'relevance'. With this measures, the first result returned by YouTube is good enough to be used in the implementation.

It is not certain whether the video contains the correct track at satisfactory quality. User feedback helps a future version of this application to identify the best matching video to a given track. Also at this time the query does not need any optimization but in future this might change, as YouTube is changing as well. In the next subsections, there is also a discussion about the query results' post-processing.

5.4 Post-processing YouTube Results

It was mentioned in the query design section that it is not necessary to optimize the YouTube query for the time being. But how about post-processing of results. Does that bring better content for the tracks?

The weighted statistics presented in the previous section is also used to analyze the effect of post-processing the results returned by YouTube. Based on this statistics, the currently implemented algorithm (that just receives the result that YouTube sets as most 'relevant') returns the manually compared results only for 61% of the tracks (Figure 5-5). This shows that YouTube finds an additional 20% of the tracks but the algorithm fails to provide them because of the lack of post-processing.

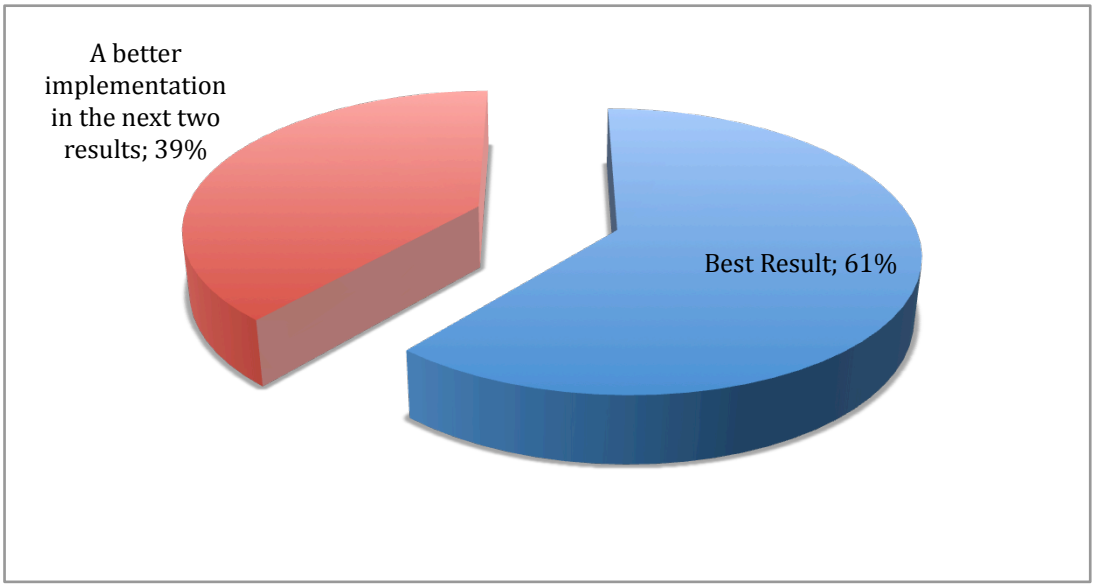

Figure 5-5 - YouTube's Relevance-Sort Analysis

The following approach shows how a few basic rules could help the implementation increase the results quality through results post-processing. The proposed rules are created based on observations when analyzing the tracks where an implementation better than the first one was among the first 3 results.

The post-processing algorithm should process only the first three results proposed by YouTube when sorted by relevance. In the following a rating mechanism is presented that, after applied to all the three results, determines the best one based on the obtained score. Note that the rating values are only first guesses that provide decent results; they have to be optimized and their performance analyzed to before being used in the application.

All three tracks have a score of 0 at begin. To this score the algorithm can add or subtract points.

Table 5-1 - Post-Processing Rating Rules

Rule		Score
Is first track in result set		+3
Is second track in result set		+1
Either title or description contains the word:	Music video	+4
	Official/original	+4
	Video	+2
	Lyrics	+3
	HQ/HD	+2
	High Quality	+2
	Mtv/bbc	+1
	Live	-2
	Tour	-2
	Instrumental	-4
	Cover	-4

In order to see the rules in action, the tracks from the statistic were analyzed again, showing the results in Figure 5-6. The percentage of not delivered tracks has lowered by 12% (from 39% to 27%) leaving only 8% of the results non-optimized. This analysis shows that the post-processing can have an important impact on the result and should be implemented in future.

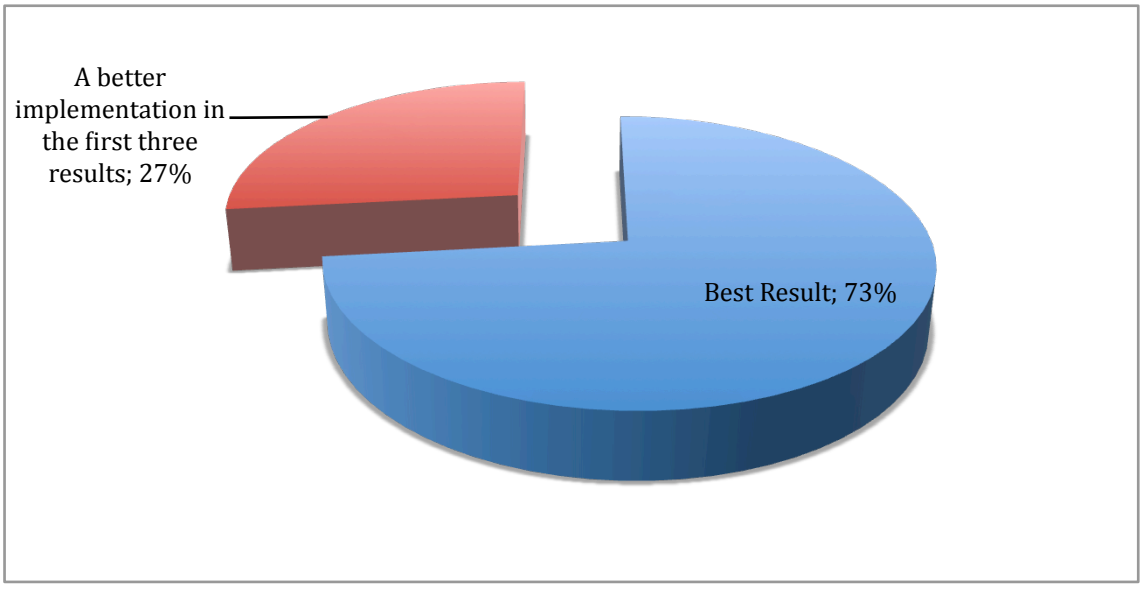

Figure 5-6 - YouTube's Track Coverage After Post-Processing

Having decided upon YouTube as content provider, the rest of the implementation details are discussed in the next chapter.

6 Architecture

After discussing the application's approach to serving its goal, it's time to describe the technical details of the implementation. What functions should the client and server side implement? How modularized can and should the application be built? What is the best approach in dealing with the huge data amount? What advantages do the different APIs bring and how can they be used?

6.1 Client and Server side Functions

The goal of the application is set and also the ground rules – also for implementation – have also been set in the Vision Chapter. The application is designed as a two-sided application: client and server side.

6.1.1 The Client Side

The client side must implement the UI of the application. It must be a stand-alone program that only requests and delivers parameterized data from and to the server. It is important to have a strict and simple communication between the two applications (client and server) such that an extension or even replacement of each one of them is easy to understand by the developer. The client application is responsible for delivering an intuitive UI and handling all requests of the user that are related to the UI. Basically the client implements the whole music player UI, with play, pause, repeat functions as well as playlist generation and modification. The list of tracks actually present in a playlist is generated by the server and returned to the client with enough parameters (video id) such that the client itself can download the music content from YouTube. No music content is available from or routed through the application's service.

The seed to be used for generating a new playlist is not selected in a traditional 'search and select candidate' manner. A type sensitive field responds to each keystroke with a suggestion list for tracks that contain the words either as artist or title. The user has to select a track from this list and use it as a seed for the playlist he/she wants to generate. The client does not fetch results from the server for each letter the user types. A delay is set such that a suggestion request is sent to the user only if the user has stopped writing for more than 'delay' time ago.

The playlist is fully customizable and offers the user the possibility to rearrange the tracks in order to influence the playing order, to remove and to jump to tracks. No restrictions are made on the rearranging of the playlist; for example if a user whishes to replay a track that he/she already heard, he/she is able to do that. Also the playlist is responsible of always having a minimal number of tracks loaded after the currently playing one. If this number decreases because of playing track advancement, track removal or reordering, the playlist automatically loads new tracks until the minimum size is reached again.

The UI has an area used to display track specific information like title, artist and other. The user uses this area also to directly give feedback about the current track (low quality or bad track choice). A direct link to the implementation on YouTube is also available.

6.1.2 The Server Side

The server must provide the following two services for the client application: playlist generation and suggestion generation. The core of the server is the only non-interchangeable part and it acts as a bridge between the modules to provide the services requested by the client. The server side application consists of modules that handle: the database, the YouTube related data, the music space data and the client connection socket. The client connection socket is distinct from the core module, such that it can also be interchangeable if a different type of client is used to connect to the same server. In this case however, the core has to be modified as well to provide the new functionalities. Also the socket has responsibilities like maintaining the session and converting the data from the structures (Objects) used in the server to the ones used in the communication to the client. Figure 6-1 - Modular Structure of the Server Side Application offers an overview of the modular structure.

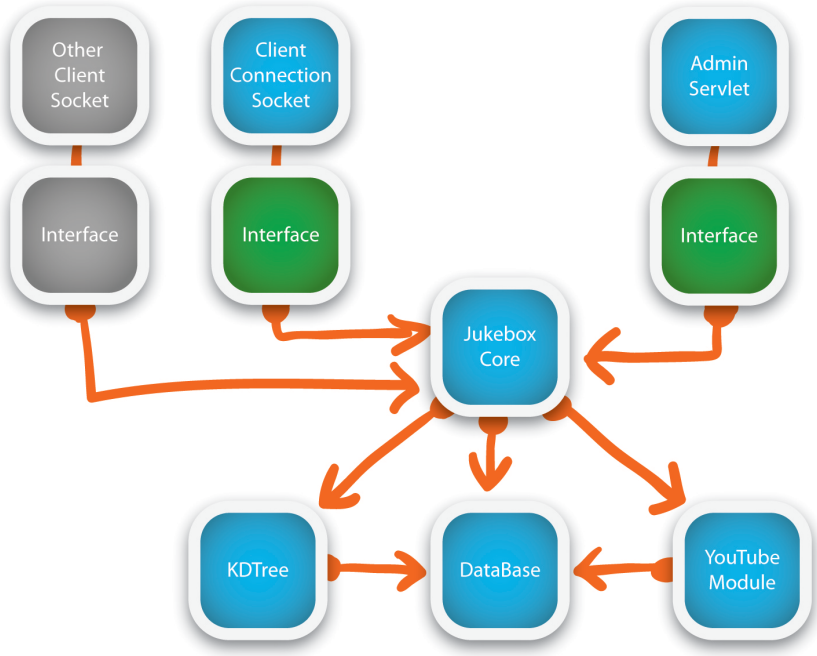

Figure 6-1 - Modular Structure of the Server Side Application

The suggestion service[7], uses an algorithm that executes an indexed crawl in the database for the words currently entered by the user. Words shorter than 3 letters are not included to avoid a high number of results. Also the results returned by the suggestion algorithm are sorted by popularity, such that the most popular tracks come up in the limited number of suggestions at an early stage of typing.

The server also offers the possibility to monitor and change its state by other means than the client application or the console. A servlet connects to the service and enquires status information like: state of the music space, state of the db connection or errors that were thrown upon client requests; this information is printed out in a web site so it can be accessed via a web browser. The

[7] Service that proposes tracks as result of user inserted words. Described later.

administration tool can be extended to implement functionality to reinitialize the server, to read and output user statistics etc.

6.1.3 The Communication

Using only standardized communication between the client and server helps future developers to interchange the components seamlessly. It's important for calls to be serviced directly, to return the requested values and to finish upon returning. Otherwise, if calls depend on local variables and states, the communication becomes very complex. The only two states the service can be in are: initializing and running. The initializing state is only at startup so it is assumed no request ever finds the server in this state (except for the administration interface).

Well-defined interfaces are at the connection of the client application to the client socket of the server; between the client socket and the core of the server and between the core and each of the modules of the server application. Several objects have been specially designed for – and are used only in – the communication between server and client. Each request is executed immediately, sequentially without (many) threads running concurrent to compute the results. Responses to requests are synchronous - always returned to the call and not sent back at a later time (when done).

6.2 Music Space (KDTree)

The music space is built of tracks that are mapped to points in a 10 dimensional field. To be able to find nodes in this space and get their neighbors, it is necessary to find a method of storing the data to be accessible in a fast way. A KD Tree (Wikipedia, KD Trees, 2009) is a right choice for this, as it is very fast at computing the nearest neighbor – O(log n) – and also nearest neighborhood – basically the two functions needed for the proposed playlist generation algorithm.

There are only few libraries that offer solutions for KD Tree implementations. One of them is Perst[8], a database that can store data in a KD Tree. As Perst is still under development, the KD Tree algorithms are not complete and there is no efficient implementation for retrieving the nearest neighbor or nearest neighborhood. The advantage of Perst was that the whole music space wouldn't have been stored into the memory, but rather in a database.

The second option that is also used because of the shortcomings of Perst, is a KD Tree fully loaded into the memory. KDTree.jar is a KD Tree implementing library for java. The disadvantage of using KDTree.jar is the huge amount of used system memory, occupied by the music space, but also the reloading process that has to run every time the server is restarted. However there is also an advantage when saving the tree in memory: high access speed. Only the coordinates of a track are saved in the tree, together with the trackID and the track's popularity. The track's title and artist together with other information is crawled from the database at run time as this occupies too much memory.

---

[8] Open source, object-oriented database – www.mcobject.com/perst/

6.3 Playlist Generation Algorithm

6.3.1 Existing Solutions

The base of the playlist generation algorithm is the music space containing data about similarity between tracks. The music space basically states that the smaller the distance between two tracks, the more similar they are. The topic of creating algorithms that use this property of the music space to generate playlists has been addressed by several ETH internal projects discussed in the Related Work Section.

The existing musicexplorer web application allows the user, in a visual 2D way, to select two tracks representing the beginning and the end of his/her playlist. The algorithm then draws a line in the music space connecting the two tracks and adds tracks found in the vicinity of this line to the playlist such that in the end the user has a playlist that represents a smooth transition between the chosen tracks. Some problems arise from using this approach, because the algorithm never knows what two tracks the user selects. If he/she just states two of his/her favorite tracks, the distance between these two is very short in music space metric, not allowing the algorithm to find optimal transition tracks. A very high distance might make the transition not smooth enough. Optimizing in respect to this variable – distance – is a difficult task and it's even more difficult to prove its correctness. Also another downside is that the generated playlist has a fixed size and cannot be extended.

Also the Amarok2 plug-in (Känel, 2008) uses a similar way of computing playlists. The difference is that it does not take only two inputs but an indefinite number, such that the user can state several of his/her favorite tracks. First, these are reordered such that a minimal path is used to reach all of them. Next, the algorithm walks along the paths with a predefined step size and after each step it chooses the nearest track to that point in the space. After reaching the end destination, it randomly chooses a point in the graph and walks to that one such that the user can explore other types of music. The algorithm is intuitive and has several advantages; the user can insert several favorite tracks and implicitly helps the application to understand his/her music taste better. The random chosen point at the end of the walk encourages the user to explore new tracks but even more, it allows the generation of an 'infinitely long' playlist.

6.3.2 First Approach

Both the approaches presented in the previous section tend to exceed the simplicity that's set as a goal of this application. Also the user might be overwhelmed when asked to insert several tracks; he/she might just have one track in mind that he/she likes to listen to.

One first approach when dealing with only one seed track is to simply take the k closest tracks to the seed. Through this, the user is likely to like the generated list of tracks. An immediate problem is when trying to generate an 'infinitely long' playlist – the transition between the tracks is not very smooth after a while. Figure 6-2 shows succession of tracks in a playlist after a certain number of tracks have been already played. You can see how the algorithm makes very big jumps to get the track that's nearest to the seed. Note that red tracks are ones

that have already been picked by the algorithm previously and cannot be chosen anymore.

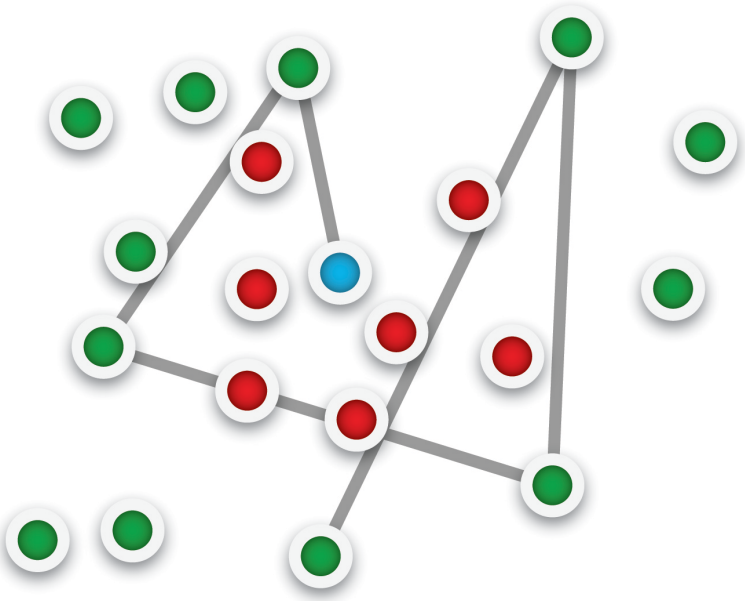

Figure 6-2 - Jumping behaviour when using static seed

A modification of this algorithm might bring the solution to this problem: reassigning the seed for each iteration (Figure 6-3). The playlist generating algorithm is a successive iteration of an algorithm that chooses the nearest track to the seed, followed by setting the newly found track as seed for the next iteration. In this way the application avoids as much as possible jumps over long distances in music space metric. In the same scenario as the last example, the modified algorithm performs without big jumps.

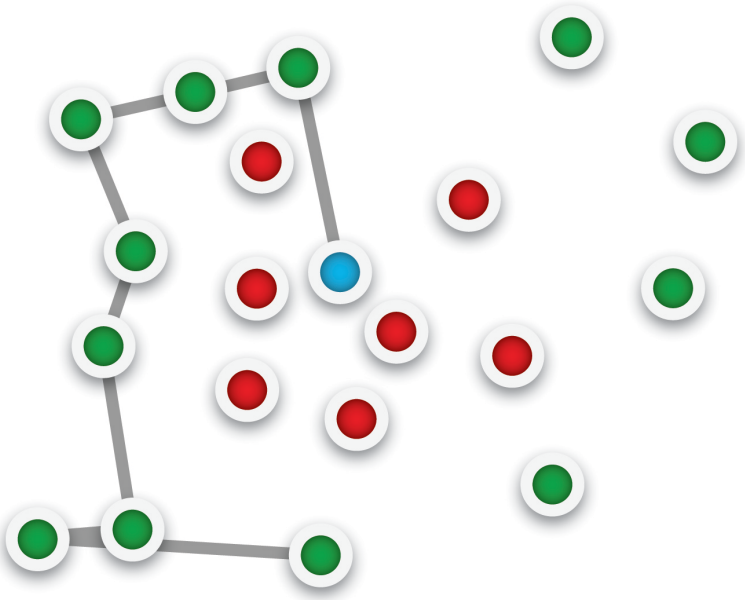

Figure 6-3 - Smoother jumping when dynamically assigning seeds

The modified algorithm also has other advantages: the distance between the chosen tracks is always very short, guaranteeing a smooth transition. Not only can the algorithm provide an infinite number of tracks, but also each track is guaranteed to be very similar to the previous one. The random walking behavior of the Amarok algorithm (Känel, 2008) is also simulated, allowing the user to randomly explore unknown regions of the music space.

6.3.3 Improvement

Choosing the nearest track to the current seed provides results equally distributed regarding the popularity of the tracks. But the probability of users enjoying the returned tracks is greater if the popularity of the returned track is maximized. An improvement suggestion is therefore not to return the track nearest to the seed, but the track in the vicinity of the node that has the highest popularity. To find it, the algorithm queries for the first 10 tracks in its vicinity and takes the one with the maximum popularity. Figure 6-4 shows a playlist-computing example with and without this enhancement.

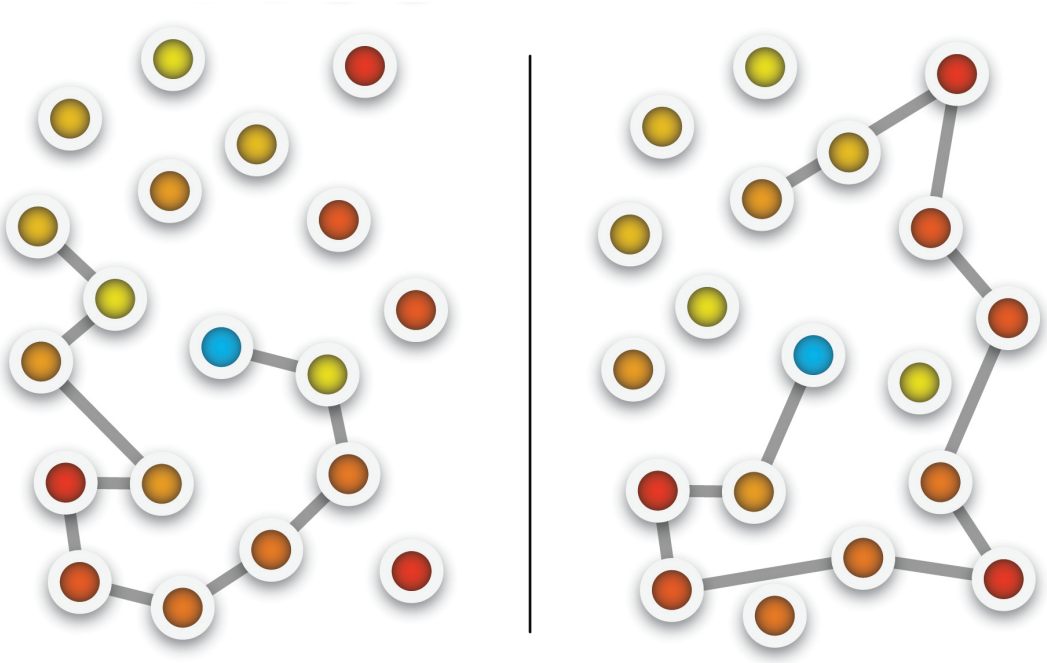

Figure 6-4 - Algorithm performance with (right) and without (left) vicinity priority search

The example in Figure 6-4 shows the algorithm performing with or without vicinity popularity search (the vicinity consists of three tracks). On the left the algorithm selects much less popular tracks in the first 10 than the one on the right; but all this comes at a cost: the step size of the right algorithm is no more than three times greater (on average) than the one on the left.

Because songs of the same artist are usually very close together in the music space, the application tends to select several songs of the same artist and place them successively in the playlist. This is an undesired behavior. To avoid it the algorithm can be taught to also avoid tracks whose artist has appeared in the last 4 tracks. By applying this additional constraint, the algorithm avoids placing tracks of the same artist less then 4 tracks apart.

When searching for tracks in the vicinity of a seed, tracks that have already been added to this playlist in the past are omitted – as a rule. Doing so, if the algorithm was not able to 'move' to a different region in the music space it starts picking the lower popularity tracks. So after 24 tracks the algorithm can start adding tracks that were present 24 tracks ago. Why 24? 24 tracks with an average of 5 min/track are almost two hours of music – enough time to re-enjoy the tracks.

6.4 User Feedback

The importance of user feedback (both indirect and direct) has been discussed throughout several sections of this report. Most important when designing the feedback algorithm was the decision what data to store so that it is of use to any future evaluation.

6.4.1 Direct and Indirect User Feedback

Direct user feedback is represented by the feedback the user chooses to submit to the application in order to help the community. Currently two types of direct feedback are implemented: 'Quality is bad' and 'I don't like the track'. Because

the application doesn't directly react to feedback (yet) this feedback is stored for a later processing. This direct feedback shows for example learn how many users actually used the feedback algorithm.

The users should hit 'Quality is bad' to signal that a YouTube video of a track has bad quality or doesn't show the track at all. In a future version, the application immediately reacts to this feedback and propose a different track or, even better, let the user propose a better track by presenting a list of alternatives.

The 'I don't like this track' is intended for users notifying the algorithm that the chosen track was not chosen right by the playlist generation algorithm, because this algorithm should only generate tracks that the user likes.

Indirect user feedback is feedback recorded and reported by the client application and without the knowledge of the user. It's important to log also the behavior and commands of the users to be able to figure out algorithm and application flaws. For example, if a user listens to the first 5 seconds of the first three tracks, but listens the whole fourth track, it means that the first three tracks were not a good choice for this user.

6.4.2 Activity and Feedback Log Structure

Because most computations of the application are done per session basis, it is good to keep track of the user's sessions and their activity in these sessions. That is why the application uses a table called tlbFeedbackSessions to save information about the session. This information includes the sessionID of the session – which at the same time is the date and time the session was first started stated in milliseconds since Jan 1st, 1970 – and the IP address of the user to compute a user-country statistic and to help user-tracking over several sessions.

All commands issued by the users are saved in a table called tblFeedbackCommands. Most of the commands issued by the user (as direct or indirect feedback) but also commands issued automatically by the application (like a track finishing playback) are recorded together with their time of issue – at the same time being a commandID, the associated sessionID, the operation code and the trackID upon which the command was executed. The list of operation codes with their explanation is found in Table 5-1 .

Table 6-1 - Operation Codes Logs

Operation Code	Command	Track ID
-1*	Report bad video	
-2*	Report bad choice	
0	A track reached the end and the next is played	The next trackID
1	Manually select a track to play	
2	Use one of the tracks as seed	
3	Manually remove a track	
4	Track lifted to be moved**	
5	Track put down after move**	trackID before which the

		dragged track was put down, or -1 if it was put down as last track in the playlist
6	Toggle repeat off	
7	Toggle repeat on	
101***	Generate Playlist Command	The seed
102***	Get more Tracks	The seed
103***	Refresh Playlist	The seed

* Negative operation codes show direct user feedback, while positive codes show indirect user feedback.

** These two commands are supported but not implemented on the client side application. Hence, if a track is moved no command trigger is issued to the server.

*** Commands having codes of the form 10* are not issued by the client side application, but by the service itself, when the user requests new tracks for example

The table tblFeedbackTracks holds some special case data. When the user issues a generate playlist command, the service logs that command in tblFeedbackCommands but also saves the tracks returned to the user as result of his/her request in the tblFeedbackTracks table. Each track is saved with the following fields:

- commandsID – representing the commandID whose result generated this trackID;
- the trackID;
- the position, representing the position of the track in the returned array + videoID.

The latter is important for example for evaluating commands like 1 – Manually select a track to play. If the user did manually select a track to play, it could mean this track is one of the users favorites so the algorithm can see where in the list it was and how the algorithm could be improved to have this track appear earlier in the playlist.

7 Conclusions

The conclusions section summarizes the projects final state and reconsiders the main ideas and results.

The goal of this thesis was to implement a user friendly, very simple to use, online music player. The approach set off by deciding which the ground rules of the implementation are. To pursue the main rule of creating an easy to use music player, an in-browser implementation has been chosen but this couldn't live without a server implementing some services the client application couldn't

handle by itself. Among these services, the most important was to generate playlists for the music player based on track similarity analysis.

The UI was the next thing to debate and how to design an UI that is both very simple and intuitive, but also powerful enough to provide enough information to the service when asking for playlists and also to be able to record the user's action for a future evaluation.

YouTube has been chosen as main music content provider and the main arguments that brought it a step in front of Last.fm were the good enough coverage of the music space and an availability of full-length tracks, whereas Last.fm usually provides only a 30 seconds snippet for a track. Analysis of YouTube coverage shown that in a real life example, YouTube offers content for about 80% of the tracks. Unfortunately, the current implementation can find only 60% of the tracks in a satisfactory quality range. However it has been shown that with post-processing of the YouTube results, an additional 17% can be mapped to good quality track implementations on YouTube, making it able to provide good results to about 77% of the user's requests. Also it has been shown that through a good feedback system, these numbers can be boosted even further. Note that these percentages reflect the real world usage statistics of the application (i.e. highly popular tracks); they cannot be generalized for the whole music space.

The playlist generating algorithm is based on analysis of the provided music space and its possibilities as similarity measurement tool. Several algorithms previously developed by ETH members were presented and analyzed both in respect to their own advantages and limitations but also their use in the context of this project. An algorithm has been developed that uses only one seed to create an 'infinite' series of succeeding tracks – meaning tracks that are each very similar to the previous and next one. The simplicity of the algorithm makes it possible to request any number of tracks from the system and the similarity metric between two successive tracks to be constant over the whole list of returned tracks. Also, the list attempts to take the user on random walks in the music space to explore new regions of music. Enhancements brought to the algorithm, like using the popularity value of each track to make a more advanced selection of neighboring tracks, raised the average popularity of the outputted tracks making the algorithm provide tracks that are even more attractive to the users.

The goal of building a simple skeleton application for providing an online music player has been achieved. The application is a good playground for further developments or innovations especially in the algorithmic part.

8 Discussion and Future Work

The Section deals with the ideas that didn't make it to the development process mainly because of the tight time schedule, but that are important to mention and at some point or another have to be addressed in future development.

8.1 Achievements

The goal of the thesis was to create a simple application to be used as a playground for future ideas of providing valuable music content to users. Also a simple algorithm was developed that delivers good playlists with minimal input from the user side.

8.2 Future Work

8.2.1 Nice to Have

Especially on the side of playlist-generating algorithms there is space for a lot of new ideas. Past ETH internal reports have shown that there are a lot of ideas that can make the best out of the given music space. Some of the ideas were discussed in this report to find a new algorithm that combines the advantages of the other algorithms while repressing the downsides. Of course no algorithm is perfect in every sense, but especially this report shows that one can find a better algorithm as other approaches, as long as it's customized on the targeted solution. Algorithms that require more elaborate user input can be easily deployed and tested on the developed application because of its module-based structure. Some examples of such algorithms are found in the other ETH internal reports like the musicexplorer web application, the Amarok plugin and a (not further detailed) like/dislike track generation algorithm (developed for the Android platform, (Bossard L. , 2008) currently under submission).

When discussing the KDTree and its all-in-memory implementation, the disadvantages of this solution have been presented. A future step is to integrate the KDTree more into a database, rather than storing the data in memory.

In retrieving the results from YouTube the application does perform well only on 60% of the tracks. A post-processing algorithm has been presented that helps the rising this margin to 77%; this algorithm has not been yet implemented in the application. Also the integration of user feedback is very helpful if considered.

Depending on the concurrent user access on the service, parts of, or the whole concept of shared resources would have to be generated all over. Especially the suggestion retrieval algorithm, currently running in the Server's core, could be externalized to a module and enhanced not to crawl the database each time a user hits a keystroke.

8.2.2 Feedback Analysis and Integration

The feedback recorded by the application is very important for it's future development. It gives a lot of information about the users behavior and allows a good analysis of their actions when using the application. Based on the results, some of the parts of the application can be changed, most probably the playlist generation algorithm if it turns out that the people are not very satisfied with the results.

Even more important is to analyze and react to direct user feedback. This is especially essential in managing the quality of the YouTube videos of the tracks. Because this quality warranty is out of the hands of this application it's important to react to the user's feedback on this matter.

Implementing a mechanism that allows users to suggest a better YouTube content based on several proposals is an important tool for the current application, because it would allow the users to directly help at providing good music content. Through implementing an advanced user feedback function, the application could be designed to analyze the received feedback and to update its state automatically and in real time, for the benefit of the whole community.

8.2.3 User Awareness

Another point only touched by the current implementation is the Facebook integration. The application is available on Facebook and users can recommend it to their friends, but the application is not using the access to the user's data yet. This data could prove very important, especially for generating the application's own – unbiased – track similarity space. Tracks listened to by users could be analyzed over several sessions, not only session-based as it happens now, moreover it could be compared to the history of friends to determine ways of recommending tracks based on the friends' preferences.

Having such a knowledge about the users could lead to completely new ways of generating playlists, maybe without any user interaction at all… similar to Last.fm's approach, but with possibly better results because the playlists could be based not only on the private but also on the friends' track history.

9 Bibliography

Bossard, L. (2008). *Pancho - The Mobile Music Explorer.* Zürich: DCG.TIK.EE.ETZH.CH.

Bossard, L. (2008). *Visually and Acoustically Exploring the High-Dimensional Space of Music.* Zürich: DCG.TIK.EE.ETHZ.CH.

Gonukula, A., Kuederli, P., & Pasquier, S. (2008). *MusicExplorer: Exploring the Space of Songs on your PC.* Zurich: DCG.TIK.EE.ETHZ.CH.

Google. (2008, 04 18). *Google Web Toolkit*. Retrieved 04 18, 2008, from Googe Code: http://code.google.com/webtoolkit/

Lorenzi, M. (2007). *Similarity Measures in the World of Music.* Zurich: DCG.TIK.EE.ETHZ.

Unknown. (2008). *Amarok Plugin.* Zürich: DCG.TIK.EE.ETHZ.CH.

Wikipedia. (2008, 04 18). *AJAX (programming)*. Retrieved 04 18, 2008, from Wikipedia: http://en.wikipedia.org/wiki/Ajax_(programming)

Wikipedia. (2009, 04 22). *KD Trees*. Retrieved 04 22, 2009, from Wikipedia: http://en.wikipedia.org/wiki/Kd_tree