Computation and Visualization of Subjective Artist Similarity for Music Libraries on Android Devices

Manuel Maly

Institute of Software Technology and Interactive Systems
Vienna University of Technology

Abstract. Abstract goes here.

Keywords: subjective artist similarity, multi-dimensional scaling, audio analysis

Table of Contents

1 Introduction

2 Related Work

In this section, the reader will be introduced to the proceedings in science which are relevant or related to this thesis. They are grouped into the following topics of interest:

- **Motivation for this thesis** - Provides an explanation on why the chosen topic of this thesis is generally of interest, and why similarity measures in music are feasible (given optimal circumstances).
- **Features of digital music** - Gives an overview of existing methods of feature extraction which are purely based on computational methods.
- **Subjective music similarity computation** - Lists literature which is related to this thesis' problem of computing the subjective similarity of artists, which is based on subjectiveness as experienced by humans.
- **Visualization of artist similarity** - Provides an overview of existing methods and models of visualization of music similarity (or, in this case, artist similarity).

2.1 Motivation for the Topic of This Thesis

Music is an integral part of the daily life in nearly all societies, and the list of published titles is growing every day. As huge amounts of data tend to be hard to digest, ontologies have to be created, by which music can be categorized in a hierarchical fashion. Aside from the author's motivation of choosing the topic of this thesis, a great interest in music classification can be observed in scientific literature. This is related to the problem that the categorization of arbitrary music titles is neither implicit nor trivial. Serving the demands of Electronic Music Distribution (EMD), the authors of [7] elaborate on the feasability of music similarity measures. It is found in [7] that the introduced similarity measure (timbre similarity) combined with other measures can yield interesting results. It is also mentioned that the interpretation of experimental results in the field of music similarity is challenging due to the subjective demands.

It is clear that even the best-educated music experts could hardly agree on any distinct similarity measure between two music titles, due to the implicit fuzziness of subjective measures. It can be assumed that it is rare that two humans would agree on the same similarity between music files if they vote independent of each other.

2.2 Features of Digital Music

As opposed to subjective artist similarity, there are music features or measures which can be retrieved by purely computational approaches. In the field of audio feature extraction, a wide range of classifiers (feature extractors) has been created. These classifiers in many cases run a bitstream analysis of a digitally stored music file and extract one or more reproducible measures characterizing the file. Interestingly, it is confirmed in [25] that the use of psycho-acoustic enhancements before feature extraction improves the classification accuracy significantly. It can be concluded that the outcomes of audio feature extraction are influenced by many factors which are not always intuitive. As has been mentioned previously, most audio classifiers analyze the bitstream of music files - however, the bitstream is only one dimension of a piece of music, if we regard it as a multidimensional object. For example, it is also possible to analyse the lyrics, as has been done in [27].

2.3 Subjective Music Similarity Computation

Subjective similarity, as the author understands it, expresses human opinions on a certain object. As previously mentioned, it is obvious that humans will hardly agree on attributes of music, and the same person might even make different statements in the course of time, depending e.g. on her mood. The following applies to both artist similarities and music file similarities, since the former may be constructed from aggregations of the latter (it has to be noted at this point that many artists tend to produce music from multiple genres, thus making an artist-to-artist-comparison difficult or even infeasible). In article [13] it is found that it is doubtable that a common ground truth for subjective artist similarity even exists, because of the inhomegeneity of measures made by the involved users. It can be deduced that a meaningful model of subjective music similarity will in most cases only resemble a compromise between different stakeholders. As inferred from [8] and [28] there are different approaches to retrieving a model of subjective similarity for a given set of music files, which include:

- Conduction of surveys with end users
- Opinions of experts
- Co-occurrence of files in end users' libraries or playlists
- Data mining of text in web sources, as performed in [40]
- Leveraging data gathered by social music services

As it is intended by this thesis to provide a concept for a fast and fully automatic approach to similarity measuring, we will concentrate on the last approach, the usage of data provided by social music services. Hybrid computation methods, such as the method described by [28] (combining acoustic features with text excerpts and tags retrieved from online services) turn out to be hardly feasible on a mobile device because of performance requirements. It is assumed by the author that for a rough estimation of music file or artist similarity, the data provided by social music services (as opposed to hybrid approaches) is sufficiently meaningful, as their daily user base is in the millions and still growing.

Apart from the source of similarity metrics, also the scope of computation has to be given thought to. Depending on a user's scenario, the user might want to explore her own music collection, or she might want to discover music similar to her own. In the following subsections, both cases are considered.

Similarity Computation for Collections of Music

In order to provide a meaningful, semantic overview of a collection of objects, data must be available or generated for all objects in the collection, or for most of them.

As has been mentioned before, **extraction of features** of music files may be used to gather metrics about the analysed files. The data retrieved may then be combined into a feature vector of N dimensions, where N is the number of features. When all files in the analysed music collection have been classified in this way, a **self-organizing map** can be trained with the resulting feature vectors, mapping vectors with small Euclidean distances spatially near to each other [37]. This results in a map where elevation represents the frequency of vectors in certain areas - thus, a high elevation at a point on the map means that the feature vector attached to the point is similar to a big number of music files. It is obvious that self-organizing maps are well suited for visually clustering music files by their features. However, it must be noted that this approach is only feasible if a huge amount of music metadata can be retrieved, either by feature extraction or other methods like text mining. Therefore, self-organizing maps can be considered not being quite suitable for the goals and scope of this thesis.

An alternative to feature extraction in this context is the construction of a **similarity matrix** containing all objects of the analysed music collection. A similarity matrix contains the similarities (or rather the dissimilarities) of all items to each other, in the form of distances. Intuitively, the measured item-item distance is higher if they are more dissimilar to each other (a distance of zero expressing equalness between items). Data sources for similarity matrices describing music include (as described earlier in this chapter):

- Surveys, playlist co-occurrences, user collection co-occurrences, web text mining,...
- Similarity rankings or measures from public Web APIs

As mentioned before, we will concentrate on the latter for complexity reasons. The task of building up a similarity matrix for a collection of music titles would then be reduced to a number of Web API calls, mapping the resulting distances or rankings to the objects in the collection. Similarity matrices can then be processed to find a suitable two-dimensional representation in Euclidean space, where the spatial distance between objects represents their similarity, or their dissimilarity respectively. This process is called **multi-dimensional scaling (MDS)**, and it has found broad adoption in scientific and industrial applications.

One of the most common multi-dimensional scaling techniques is the adoption of a spring model, which emulates the physical behaviour of steel rings connected by metal springs [31]. To continue with the analogy, MDS starts off with the steel rings at random positions, and in multiple iterations tries to satisfy the springs' forces. A spring's force is usually proportional to the discrepancy of two objects' high-dimensional distance and their low-dimensional distance.

The fitness of the model (all steel rings being at their optimal position, considering all connected springs) is described by a stress function. The lower the stress function's output, the better the new low-dimensional model suits the original high-dimensional model. In most cases a perfect match between high-dimensional and low-dimensional Euclidean distances (stress function output = 0) is not possible, and thus for the MDS calculation to terminate, sensible termination criteria have to be defined - e.g., if the velocity of objects moving at each iteration falls below a predefined threshold, the algorithm terminates and the resulting low-dimensional representation is accepted as being "good enough".

Unfortunately, common spring model (or: metric distance) MDS is inflexible in the sense that the whole computation has to be performed all over again if slight changes to the data set occur [31]. Therefore, a computationally advantageous and more flexible approach to multi-dimensional scaling has been presented in [31], combining **MDS with sampling and interpolation**. As opposed to common multi-dimensional scaling, this hybrid methodology starts off with a random sample of size $\sqrt{N}$, where N is the number of objects contained in the dataset. After the spring model computation described above has been performed on the subset, the rest of the data set ($N - \sqrt{N}$ objects) is integrated into the low-dimensional model by an interpolation process which is described in detail in [31]. It has been found in the article that this combination of algorithms improves greatly on the accuracy of the model (i.e., a lower stress function output) and offers a sub-quadratic run time of $\mathcal{O}(N * \sqrt{N})$.

Discovering Objects Similar to a Given Object

After the elaboration of means of exploring the semi-static collection of objects in a user's library, heed must be given to the recommendation of unknown objects. Equipped with similarity data retrieved from various web APIs it is not only possible to compute relations of objects within a library, but also to find related objects which are currently not present in the library. It is clear that the same algorithms which have been previously described would compute usable outcomes by simply adding unknown objects to a library's representation (e.g. spring model MDS); yet, it can be assumed that in most cases only one object in a library is the starting point of a search for similar objects. This renders a big part of the library's representation irrelevant for this use case - consider a user searching for interprets similar to "The Beatles" - most likely, she will not be interested in how these unknown interprets relate to other bands in her library. Also, integrating such previously unknown objects into the representation of a big library will be computationally and query-wise infeasible, as dissimilarity distances to all objects in the library have to be determined. It must be noted that the representation for unknown object recommendation can only be a crude approximation (because of the previously described volatility of subjective similarity), and for this reason not only numerical measures, but also similarity rankings are considered sufficient for this use case.

A derivation from a previously proposed method can be considered here: Suiting the requirements well is a spring model MDS approach applied to a small dataset consisting of the starting point object (in the example being "The Beatles") and previously unkown objects which are most similar to it (retrieved via web APIs). The size of such a dataset would presumably peak at 30-40 objects, making common spring model multi-dimensional scaling computationally feasible. Naturally, the sampling/interpolation approach described in the previous subsection would also apply here, further decreasing the computation time.

However, a computationally less expensive algorithm for such means has been proposed in [26], consisting of a fusion of similarity rankings from various social music services. In this article it is demonstrated that various methods of embedding (fusing) similarity rankings from online services can provide different meaningful similarity models, some of which give more weight to unknown artists. Intuitively, this methodology is able to compute a ranked list of similar objects, based on multiple sources for greater reliability, in a customizable way. The fusion methods reach from rank-average to concordet-fusion (unweighted directed graph). Three major benefits speak in this approach's favour over global numerical similarity measurements:

- **Potentially insignificant computation times** while preserving a stable similarity ranking very well suited for mobile end users.
- **Simple but effective customization** achieved by easily exchangable fusion algorithm components.
- **Reducing the number of web API queries to a minimum** greatly reduces the overall number of network requests, making the method even better suited for mobile usage.

It must be noted at this point that the rank fusion algorithm is neither able to define distances between arbitrary objects in the dataset - only a dissimilarity ranking between the starting point object (e.g. "The Beatles") and the remaining objects is obtained - nor is it able to force the inclusion of objects (from a local library) in the ranking. This algorithm depends fully on the objects provided by external sources, meaning that if the starting point object is not contained in external sources, no meaningful result can be obtained. However, the author considers the amount of objects which can be obtained from third party web APIs sufficient for the algorithm to perform well for most of all objects in a typical user's library.

2.4 Visualization of Artist Similarity

The mode or fashion of data visualization can be considered a crucial aspect of interfaces for humans (i.e., graphical user interfaces). Today, humans' ability to apprehend information presented to them is limited in several ways, some of which are physical, and some of which are of psychological nature. Some of these limiting factors include:

- Restriction of short-term memory,
- Limited power of concentration,
- Narrow attention span (especially while using mobile device),
- Color blindness,...

Therefore, it is desirable to give heed to the choice of visualization method to achieve optimal apprehension results, without hindering information understanding through visualization errors. Since the field of music and music collection visualization is broad and not all algorithms can be presented within this thesis, the author decided to select only the field of two-dimensional collection visualizations for further investigation. It must be noted that several fields of visualizations are left out of the scope here, including:

- **Abstract visualization as an artform** - Certain artforms try to make music more tangible by creating matching images, as in the movie "2001 - A Space Odyssey" by Stanley Kubrick, or in works by the demo scene in Germany [39].
- **Realtime computed images as abstract visualization** have become common components of many desktop audio players, presenting the user with animated images (fractals, 3D-animations,...) which somehow resemble certain features of the currently played music.
- **3D environments resembling music content** give users the ability to roam through a virtual space similar to the way they interact with the physical world, as described in [11].

The scope of this thesis confines itself to the visualization of music tracks as objects, relating these objects to each other, and disregarding their real-time aspects (i.e., not generating any visualizations during playback). Intuitively, the computation and visualization of those relations (also, the quality of relations, e.g. ranking or dissimilarity distances) are closely related to each other. In some cases, a certain mode of computation of object relations more or less forces or forbids the usage of certain visualization approaches. Therefore, the presented modes of visualization are at least closely related to their computational counterparts from the previous subsection.

Visualization of Collections of Music

Previous work has shown that **self-organizing maps (SOM)**, which are in this context also called "islands of music" are well suited as visualizations of related music objects [10]. This methodology depends on raw audio stream analysis (performed by aforementioned feature extraction algorithms), and subsequently displaying them on an elevation-map, similar to a geographical map. Proof-of-concepts have been successfully implemented, as has been demonstrated in [33], featuring the PlaySOM. The information such self-organizing map visualizations want to give the user is: There are clusters of similar pieces of music in the provided collection, and within one cluster the contained music files most similar to each other. Additionally, each cluster has its own weight vector which can be used to add semantic height annotation to the map - for example, clusters whose objects contain a high tempo can be marked "high" (as opposed to clusters with slower music being marked as "low"), generating a corresponding height profile.

Another broad group of (dis-)similarity visualizations is made up of **force-directed graph layouts**. They all consist of nodes (music objects) and edges (relations between objects). Additionally, the distances (edge lengths) between nodes approximate a function over the previously determined dissimilarities between the music objects. As has been described in [17], the application of pseudo-physical forces on an undirected graph provides for a improvement of the graph layout. This is achieved by adding attractive or repulsive forces to all nodes in the graph, such that nodes push away from or attract each other. As long as there is energy left in a graph (i.e., there are objects which are not in their optimal position), the nodes are moved in a way that satisfies the applied forces. The authors of [32] have described and experimented with several graph-based layouts, and among them was a force-directed layout algorithm called LinLog [34], which has been found to deliver the most aestethic results. However, a computational model which might be more suitable for the calculation on mobile devices is presented (among other methods) in [23].

The forces in a force-directed graph layout can behave like springs connecting nodes, and for this reason a subset of force-directed graph layouts is called **spring model**, which has been described in length in the previous subchapter in the context of multi-dimensional scaling (MDS). The calculation of a spring model's layout is very tightly coupled with the overall MDS computation, even in hybrid approaches [31] - in the case of MDS in combination with spring models, the visualization approach cannot be cleanly separated from the computation approach.

Other graph layouts which pose options for music collection browsing include [32]:

- Principal Component Analysis (PCA) layouts
- Tree map layouts
- Space filling curve layouts

Visualizations for the Recommendation of Unknown Objects

As has been discussed in the previous subchapter, the computational approaches for the recommendation of new objects can be either very similar to the computation of ordinary collection visualization, or they can use more simplified rank-based models. The former are clearly covered properly by the broad range of previously described visualization methods for collections of objects. On the contrary, visualization possibilities for rank-based computation models are not as manifold, due to the fuzzy dissimilarities between objects - a ranking can not be used to acquire deterministic object distances. However, since the goal of the visualization of such a ranking is to provide a very rough overview to the user, a deterministic visualization is not necessary. It can even be considered to omit the ranks, and just display these objects as relations of the same relevance, as has been done by the authors of [38]. It seems that also for this use case, a force-directed graph layout provides for the most aestethic results [16]. Additionally, such layouts can execute their self-optimization in realtime while being presented to their user without affecting the user experience in a negative way, as is shown by [1]. Additionally, the nodes in such layouts are user-manipulable in realtime.

2.5 Summary of this Section

3 Scenario and Scope of this Thesis

In this chapter, the scope of this thesis will be defined and a user scenario outlined.

3.1 Scope Definition

The scope of this thesis is defined by the following goals:

- Verification of the selected artist similarity computation method being feasible for mobile end user devices.
- Verification of the selected artist similarity visualization method being a sensible choice for mobile end user devices.
- Description of the implementation of a prototype (able to perform the selected computation and visualization methods) on the Android platform.
- Description of the design and presentation of the results of a user study.

3.2 Selected Artist Similarity Computation

After consideration of the options for similarity computation in section 2, the author has concluded that the approach presented in [31] (combining multi-dimensional scaling with spring models and interpolation), seems to be a promising approach to the problem of music library visualization, accommodating mobile devices by especially fast computation. The approach will not be applied unaltered, but modifications and enhancements will be made and described in this thesis. It must be noted that this computational method is limited in the amount of objects which can feasibly be displayed (and computed) on a mobile device. Also, since for some data structures no similarity metrics are currently available, the adapted MDS method is not suitable for all kinds of music data. Therefore, a fallback algorithm is selected for the display of hierarchical data objects: the force-directed layout algorithm presented in [23] is of low computational complexity, and its results seem promising.

3.3 Selected Visualization Computation

Since an MDS computation approach has been selected for further proceeding, the visualization computation is entangled with the similarity computation and cannot be freely chosen. Also, the presentation space for visualization is chosen to be two-dimensional, since three-dimensional visualizations are hard to implement and are unlikely to be displayed as intended on mobile devices (with many objects being presented). Thus, the visualization method for MDS-generated object clusters is constrained to be a two-dimensional layout algorithm, positioning the laid-out objects such that they resemble their MDS-generated coordinates. For ease of navigation, zooming and panning will be added, and different colors will be used for faster identification of object types.

3.4 Selected Algorithm for Removal of Overlapping of Artists

The author of this thesis decided to use an iterative approach to the removal of graph node overlappings - a modification and simplification of the idea of the force-transfer algorithm [21]. Graph nodes are added into a fresh space, one by one, and while they are added, they are positioned such that overlappings with other nodes are eliminated. This elimination is performed by moving the freshly added node on the vector [overlapped node's center TO added node's center] so far that they don't overlap anymore. As this might create new overlappings, several iterations of this "pushing away" are performed, until the freshly added node does not overlap any other nodes. The outcome of this algorithm is a cleanly laid out representation of the previously crowded MDS computation outcome.

3.5 Selected Layout Algorithm for Artist Discovery

Since the graph of discovered artists is not star-shaped (one single subject artist circled by discovered artists), but has multiple main nodes (potentially multiple subject artists circled by their discovered artists), it would be too complex to calculate artist positions deterministically. Instead, a force-directed layout algorithm is chosen, where all graph nodes push away from each other, but the secondary nodes (discovered artists) are attracted by their primary node (subject artist). These forces are applied continuously in iterations, as long as a certain movement threshold has not been undercut. With only one subject artist, such a graph will be star-shaped, with the discovered artists circling the subject. After the optimal parameters for these forces have been found, such an algorithm can handle an arbitrary number of displayed artists.

3.6 Overview of the Assembly of Algorithms and Information Flow

To give the reader a better picture of the sections to come, an overview shall be given which explains how the selected algorithms process and pass on information to each other. An in-depth explanation of these information flows will be given in the section 6.

Library Visualization

DURCH GRAFIK ERSETZEN:

- Extraction of music metadata on the device
- Matching of the device's music metadata with metadata from web sources
- Querying of Artist Similarity data from web sources
- Completion of Artist Similarity data
- Laying out artists in 2D space with a Multi Dimensional Scaling (MDS) algorithm
  – Building up of a distance matrix between artists
  – Generation of a subset of artists and laying them out according to spring model forces
  – Addition of the remaining artists, positioning them around the initial subset
  – Application of spring model forces on all nodes for a few iterations
- Removal of overlapping of artists' depictions in 2D space
- Display of the laid out artists in OpenGL
- Continuous reaction to user actions (zooming, panning, tapping)

Artist Discovery

Artist discovery is initiated by the user selecting a certain artist ("subject"), and requesting discovery mode.

DURCH GRAFIK ERSETZEN:

- Querying of the artists most similar to "subject" from web sources
- Integration of the retrieved artists around "subject" in 2D space, at randomized but similar positions
- Continuous re-arrangement of the retrieved artists based on a force-based layout algorithm (also reacting to newly added similar artists)

3.7 Summary of this Section

4 Computation of Artist Similarity Based on Webservices

In this section, the computation of similarities between artists within collections of music will be presented.

- **Matching of music entities from Different Sources -**
- **Multidimensional Scaling Algorithm -**
- **Optimizations for Execution on Mobile Devices -**
- **Downsides of MDS in this Context**

4.1 Matching of music entities from Different Sources

In order to use similarity data from different information sources (here: web services), these data items have to be consolidated in a way that assures that they don't mix up - e.g., objects with the same name are assumed to depict the same artist (typos or naming variations should be amended).

The following approach has been chosen to perform the gathering and matching of different online sources of similarity data:

1. A list of music files residing on the mobile device is compiled, and from there a distinct list of locally available artists is generated.
2. For all locally available artists, the available online APIs are queried, returning lists of similar artists.
3. Within the returned lists, the locally available artists are searched for - if there is a match, a similarity connection is stored for the related artists.

The matching itself takes place implicitely in steps 2 and 3. In step 2, the matching is performed by the web API itself (which is assumed to be highly optimized and reliable); in step 3, the matching happens in the form of a case-insensitive equality-assertion (artist name equals artist name).

The result of this approach is a number of similarity connections between locally stored artists. It can be assumed that as much as 90 percent or more of potential similarity connections cannot be established based on the web APIs' results - currently, they cannot be queried specifically for the similarity between artist X and artist

Y, but this information is only available when artist Y is in artist X's similar artists list, or vice versa. Therefore, a meaningful default similarity for unknown similarities must be defined. The author has decided on a statistical approximation - any artist Y which is not in artist X's similarity list (returned by a web API) is assigned a similarity connection {X, Y} of
*0.5 * [last similarity value in the list of artist X's similar artists]* (expected value).

4.2 Multidimensional Scaling Algorithm

About Multidimensional Scaling
Multidimensional scaling (MDS) means the translation of objects in a high-dimensional space into another space of more or less dimensions. Most often, MDS is used to map objects from a very high dimensional space into a two- or threedimensional space, in order to make their relative positions to each other comprehensible to humans. Dimensions in this context do not have to be spatial, or even continuous - a dimension in this sense can be any attribute which all objects in the observed set have in common (or which can be assigned to all objects). In order to eliminate or aggregate dimensions (as must be done to reduce dimensionality), several techniques have been found. One of them, and the most promising for the objective of this thesis, is the adoption of a spring model, which has been described in section 2. The emulation of a system made up of springs connecting rings of steel is well suited for the translation of high dimensional objects into a euclidean space.

Fitness of MDS for the given problem
The subjective similarity of music, or artists in particular, can be expressed as attributes which are assigned by human beings (their opinions). Such attributes may be:

- Rhythm
- Beats per minute
- Mood
- Popularity
- Genre
- Tuning

– ...

Whatever the assigned attributes are - by comparing attributes of music titles to those of other titles, a similarity measure can somehow be obtained. Every attribute then can be seen as another dimension, and every music title or artist is an object in a space of N dimensions, where N is the number of attributes assigned. Therefore, the problem of laying out music objects in two-dimensional layouts with respect to their subjective similarity can be reduced to a multidimensional scaling problem. Since the previously mentioned web sources provide readymade similarity measures between artists, we can take a shortcut here, and remain oblivious of audio attributes such as rhythm or mood. Thus, a very complex part of spring model MDS - the filtering and mapping of different attributes w.r.t. their impact on object similarity - is already implicitely performed by those web sources. Intuitively, the retrieved similarities can be used to calculate the length of the connecting springs.

Concrete MDS Algorithm

As mentioned before, the chosen MDS algorithm is based on a spring model. In the real world, steel rings interconnected with springs strive to form an equilibrium, where all forces enacted by the springs are balanced. In a perfect equilibrium, the position of every steel ring is a compromise of all forces acting on it, and the rings will not move anymore. The "system stress" is then zero. The real-world spring system will also produce non-perfect ending states because of energy loss in the system - at some point, the rings will not move anymore because the original kinetic energy has transformed into other forms of energy and is not available for movement of rigid springs. In a digital emulation of this concept, objects are moved around by high-dimensional forces (springs) until, theoretically, an equilibrium has been reached. However, the balancing effect (swinging) produced by these forces cannot happen continously as in the real word. Instead, the swinging spring effect is approximated by computing iterations, each representing a system state at a certain time. Since there is no loss of energy in the system (as observed in the real-world spring system) for every iteration, there are configurations in which the objects will be pushed around by the forces forever, never reaching a final static state. It is therefore common to limit the amount of iterations or regard the positions of objects as "good enough" when the system stress has fallen below a threshold. Since system stress grows proportionally to the number of objects (if stress among them stays the same), an optimal system stress threshold is non-trivial to calculate. Research has produced both basic and more refined algorithms which approximate the behaviour of a spring model. The most basic approach is mentioned in [9] (and then refined for faster computation) - it starts out by assigning coordinated in low dimensional space (2D or 3D is most likely) for each object (coordinates may be related to high dimensional coordinates, or may be randomized). Iterations are then performed on the low dimensional model, by performing these steps:

1. For every object, calculate its supposed low dimensional distance (LDD) to every other object. The supposed LDD is determined by a custom function (e.g., the Euclidean distance). Here, this function takes a similarity value retrieved from web sources as a parameter.
   – If the supposed LDD does not match the actual current LDD, calculate a force into a certain direction (vector), either attractive or repulsive, and save it for later application.
2. Apply all previously saved forces onto the corresponding objects.

4.3 Optimizations for Execution on Mobile Devices

The most basic MDS algorithm described previously is computationally too expensive to perform with a reasonably large data set (up to 1000 objects) on a mobile device, at a complexity of $O(N^3)$ or $O(N^4)$ [9]. Chalmers therefore introduces a stochastic sampling method and the use of neighbor sets in [9]. Instead of performing force calculations for every object to every other object in each iteration, subsets of objects are picked and used randomly. Also, for every object a neighbor set is created which keeps objects of the lowest high-dimensional distances. It is reported in [9] that these optimizations reduce the algorithm's complexity to $O(N^2)$.

Further on, [31] introduces another optimization: Initially, only a subset of all objects is selected and the aforementioned stochastic layout algorithm is performed on those. Then, the rest of the objects is added to the layout following an estimation of their best positions. Finally, the stochastic layout algorithm is performed again, for a limited number of iterations. Thus, the complexity of the algorithm is further reduced to $\mathcal{O}(N * \sqrt{N})$.

The optimized algorithm is composed of these steps (C1, C2, C3, C4 being predefined constants):

1. Select a random subset of $\sqrt{N}$ objects (N = number of all objects)
2. Perform a sampled MDS calculation on the initial layout, by performing C1 iterations of the following:
   – For each object in the subset (called "subject") do:
   –– Generate a random subset of all objects containing C2 objects; concurrently, switch objects into subject's neighbor set if either the set has not reached its capacity, or if the object has a lower high-dimensional distance than the current most distant neighbor.
   –– For every object in either the random subset or the neighbor set, calculate its supposed low dimensional distance (LDD) to the subject. The supposed LDD is determined by a custom function (e.g., the Euclidean distance). Here, this function takes a similarity value retrieved from web sources as a parameter.
   –– If the supposed LDD does not match the actual current LDD, calculate a force for subject into a certain direction (vector), either attractive or repulsive, and save it for later application. (Note: Only the subject is moved)
   – Apply all previously saved forces onto the corresponding subjects.
3. For every object (called "subject") NOT in the initial subset do (subset Z containing all objects from the initial subset):
   – Find the most similar (lowest high-dimensional distance) object in Z, and choose the best quadrant around it, to place the subject there.
   – Improve on subject's new position by performing C3 iterations very similar to step 2., but only with the initial subset of objects.
   – Add subject to Z.
4. Perform a sampled MDS calculation like in step 2. with C4 iterations, but with all available objects instead of only the initial subset. The pupose is to move grossly mispositioned objects to

a better position - very good results can be achieved with a low number of iterations (C4).

4.4 Downsides of MDS in this Context

The advantages of MDS have been described sufficiently in this section. However, there are some downsides to this approach too, some of which are inherent to the problem at hand and which could not be alleviated by using another method.

If the internal structure of the transformed music entity set **is not expressable in a two-dimensional space without tradeoffs**, the spatial distances between entities will not always depict their similarity correctly. The larger the transformed set is, the more will the entities' 2D distances be different from their expected ("should be") distances. It can be viewed as a given that in most cases distances between some entities will not at all match their expected lengths.

The layout produced by the aforementioned MDS method **will never be the same for two different calculations**, since many parts of the algorithm involve randomness. While the addition of some entities is possible after the layout has been completed, a complete rerun of the algorithm will become necessary if similarities change. This is suboptimal for users because they have to reorient themselves after every major layout change.

The MDS method presented earlier in this chapter relies on definite high-dimensional vectors (from which similarity measures can be derived), which is not available here - in the context of computing similarity based on web sources, it is unlikely that a similarity measure can be found for every entity-entity relationship. Therefore, **estimations have to be used for unknown similarity relationships**, potentially polluting the calculation with incorrect values.

4.5 Summary of this Section

5 Visualization of Artist Similarity

In this section, the visualization of objects whose positions have previously been computed (see section 4) will be discussed. Also, the interaction patterns for the mobile application prototype will be defined.

INSERT SUBCHAPTERS LIST HERE

5.1 Visualization

After the computation of similarity measures and the resulting layout of music objects has been performed (see section 4), the mode of visualization has to be defined. As has already been discussed, the possible modes of information display have been narrowed down due to several constraints, determining that the visualization must be two-dimensional. Small screens on mobile devices introduce a further constraint: in most use cases, not all of the content will fit on one screen; therefore, a zooming and panning mechanism will be employed. To provide users with a rich experience, the objects shall not be shown as points, but as shapes - if an image of the object exists in an online source, it shall be used, or otherwise, an arbitrary rectangle shall be shown. In order that users can orient themselves better, a non-solid background will be used - ideally, it will emulate 3-dimensionality by moving and zooming while the user pans or zooms the graph.

5.2 User Interaction

Information will be displayed to the user in a hierarchical way - to limit the amount of information displayed all at once, it is split up and logical links are established. The hierarchy is made up of:

– **Artists** - images and labels of the respective artists are displayed. Their positions relative to each other depict the artists' similarities. Here, the user can select an artist and proceed to the next hierarchical level by pressing a button:
– **A certain artist and her album(s)** - images and labels of the previously selected artist and her albums (stored on the device), arranged in a starlike way (without any similarity information

conveyed). Again, the user can select an album and proceed to the next hierarchical level by pressing a button:

- **An album's tracks** - a list of tracks, carrying the image of the belonging album. The user can choose to start playback of one or all of the tracks.

To improve the user's understanding of the current navigational status, so-called breadcrumbs. Breadcrumbs achieve "visitor location awareness in a simple and direct way" [36] if applied to 2D spaces. A breadcrumb in this context is a series of links displayed on top of the graph, each representing one hierarchical level. An example for breadcrumbs shown while a list of tracks (lowest hierarchical level) could be: "Artist: Air ¿ Album: Talkie Walkie". Further, a button is added which positions the viewport at the center of the graph, for the user to recover in case she has lost track of the viewport's position.

Within the graph viewport, interaction will be a mix of touch based interaction and hardware buttons, since most Android devices provide those affordances. Special accessibility functions for interaction without touch gestures are omitted since they would go beyond the scope of this thesis. Following the established conventions of touch interfaces, navigatable objects (like artists or albums) are made touchable. Likewise, the user can reach upper view hierarchies by pressing the hardware (or software-emulated) back button common to Android devices. Exploration of the graph is performed by two common touch gestures, namely pinch-to-zoom and one-finger-panning. To zoom, the user places two fingers on the graph viewport and moving her fingers either apart (zooming in) or towards each other (zooming out). To pan, the user places one finger on the graph and moves it around as if it were a piece of paper and the screen an aperture showing the paper.

5.3 Removal of Node Overlapping

As the reader may have noticed, the positions of objects computed as described in section 4 don't respect any aesthetic criteria. [24] gives an overview of such criteria, three of which also apply to graphs produced by MDS:

- Minimisation of the area taken up
- Minimisation of total object distance
- Aspect ratio close to or matching the specification

However, an even more important aesthetic factor in the perception of graph drawings is the **removal of node overlappings**. The computational layout method applied in this thesis - spring model MDS (see section 4) - can not give any heed to overlapping nodes (or margin between them), since nodes are positioned as points, not 2D objects. However, the objects do need to be displayed as 2D shapes (e.g. as rectangles, circles, images,...), and therefore the resulting layout can (and in many cases, will) contain overlappings. As this may impact the perception of the graph by humans in a negative way (objects may even be completely hidden by other objects), a way has to be found to eliminate overlappings as reliably as possible, without affecting the conveyed information of object similarities too much.

The authors of [30] therefore define the concept of the Mental Map, meaning that a graph has properties which should be maintained if transformed: Orthogonal Ordering, Proximity Relations, and Topology. An algorithm called Force Scan (FS) is then proposed to remove node overlappings while preserving the Mental Map of a graph. FS operates by first moving nodes horizontally, maintaining their horizontal ordering, and then moving nodes vertically, maintaining their vertical ordering. Several improvements to FS have been discussed ([19] [21] [24]), while the algorithm in [21] - **Force Transfer (FT)** - seems to be the most suitable for this thesis' use case, as it is of lower computational complexity than FS and produces pleasing layouts.

The FT algorithm identifies clusters of overlapping nodes, and then performs transfer scans on them: left-to-right, right-to-left, upside-to-downside, and downside-to-upside. During the scans, the nodes in each cluster are moved such that the clusters become free of overlaps. Multiple iterations may be necessary, as the separation of one cluster may cause nodes to move on top of nodes from other clusters - thereby forming new clusters. Finally, the layout eventually is overlap-free, preserving the aforementioned mental map of the graph.

5.4 Summary of this Section

6 Implementation of Artist Similarity Visualization for Android Devices

6.1 Assembly of Algorithms and Information Flow In Concrete Implemention

In section 3, an overview was given of how algorithms work together in the system. In-depth implementation-specific details of this assembly and information flow will now be given.

Library Visualization

- Extraction of music metadata on the device

  Input: Access cursor to a data store containing the device's music library metadata.
  Algorithm: Iterates through all artists and tracks in the device's music library. If an entity has not yet been registered in the app's database, parts of its metadata are compiled and put into the database.
  Output: Local music metadata stored in the app's database.

- Matching of the device's music metadata with metadata from web sources

  Input: Local music metadata stored in the app's database.
  Algorithm: Iterates through all local music metadata previously retrieved, and calls Last.fm's RESTful API to find matching entities - picking the first match as best match. These pieces of remote metadata are then stored in the app's database.
  Output: Remote music metadata stored in the app's database.

- Querying of Artist Similarity data from web sources

  Input: Local music metadata and matched metadata from remote web sources stored in the app's database.
  Algorithm: Iterates through all local and remote music metadata previously retrieved, and calls Last.fm's RESTful API to get the 100 most similar artists for each.

Output: Artist similarity relations stored in the app's database.

– Completion of Artist Similarity data

Input: Artist similarity relations stored in the app's database.
Algorithm: Iterates through the previously retrieved artist similarity relations, and adds approximations for similarity relations which have not been found in the web API's results. These approximations are calculated as described in section 4.
Output: Complete artist similarity relations stored in the app's database.

– Laying out artists in 2D space with a Multi Dimensional Scaling (MDS) algorithm

Input: Complete artist similarity relations stored in the app's database.
Algorithm: Applies a multistep (see sub-steps) algorithm which uses previously retrieved similarity data to position nodes resembling artists without overlappings.
Output: Graph strucure of nodes resembling artists, laid out such that their position indicates their similarity to each other.

– Building up of a distance matrix between artists

Input: Complete artist similarity relations stored in the app's database.
Algorithm: Iterates through all artist similarity relations, calculates a distance value ($d = Similarity * -1 + 1$, $d \in [0, 1]$)
Output: Distance matrix of artists, based on their inverted similarities.

– Generation of a subset of artists and laying them out according to spring model forces

Input: Distance matrix of artists, based on their inverted similarities.
Algorithm:

Output: Graph structure of nodes resembling artists (correctly positioned subset)

– Addition of the remaining artists, positioning them around the initial subset

Input: Graph structure of nodes resembling artists (correctly positioned subset)
Algorithm:
Output: Graph structure of nodes resembling artists, many of them suboptimally positioned

– Application of spring model forces on all nodes for a few iterations

Input: Graph structure of nodes resembling artists, many of them suboptimally positioned
Algorithm:
Output: Graph strucure of nodes resembling artists, laid out such that their position indicates their similarity to each other.

– Removal of overlapping of artists' depictions in 2D space

  Input: Graph strucure of nodes resembling artists, laid out such that their position indicates their similarity to each other.
  Algorithm:
  Output: Graph strucure of nodes resembling artists, laid out such that they don't overlap each other and their position indicates their similarity to each other.

– Display of the laid out artists in OpenGL

  Input: Graph strucure of nodes resembling artists, laid out such that they don't overlap each other and their position indicates their similarity to each other.
  Algorithm:
  Output: OpenGL view object and auxiliary system objects

– Continuous reaction to user actions (zooming, panning, tapping)

Input: OpenGL view object and auxiliary system objects
Algorithm:
Output: -

6.2 Structure of the Application

About Android Apps

Android is a mobile operating system developed by the Open Handset Alliance [5], led by Google. Its architecture allows for 3rd party programs (called "apps") to easily be run and debugged on Android devices. Android apps are run in the Dalvik VM which makes use of a register-based architecture, relying on a Linux kernel for low-level functionality [12]. The most wide-spread programming language for building Android apps is Java, but various other languages such as Scala or even scripting languages like Groovy or Lua can be used. Since mid 2009, developers can also write and integrate native C and C++ code by making use of the Native Development Kit (NDK).

From an app developer's perspective, the frameworks contained in Android dictate a user-centric application structure, made up of so-called Activities [22]. Every Activity encapsulates a screen which is presented to the user. Activities are loosely coupled, allowing only serialized objects and primitive values to be passed between them.

User interface composition in Android is performed partly in the CPU (e.g., in Java code), and partly in the device's GPU (by using the OpenGL interface). Apps can also choose between these composition variants.

Android is a strictly touch-centric operating system, meaning that most user interactions are performed via the device's touch screen. Originally, Android devices were bound to provide hardware buttons, but starting with Android 4.0, those buttons are gradually replaced with software buttons (displayed on the touch screen).

6.3 Retrieval of Similarity Data

6.4 Implementation of Multidimensional Scaling

6.5 Implementation of Removal of Node Overlappings

6.6 Visualization Details

6.7 Summary of this Section

7 User Study

7.1 Hypotheses

7.2 Experiment Setup

Population

Tasks

Metrics

7.3 Evaluation and Analysis of Study Results

7.4 Summary of this Section

8 Conclusion

References

1. http://audiomap.tuneglue.net/
2. http://developer.echonest.com/docs/v4/
3. http://devzone.wiki.meemix.com/index.php?title=MeeMix_API
4. http://www.lastfm.de/api
5. http://www.openhandsetalliance.com/
6. Agarwal, S., Wills, J., Cayton, L., Lanckriet, G., Kriegman, D., Belongie, S.: Generalized non-metric multidimensional scaling. In: AISTATS. San Juan, Puerto Rico (2007)
7. Aucouturier, J.J., Pachet, F.: Music similarity measures: What's the use? In: Ircam (ed.) Proceedings of the 3rd International Symposium on Music Information Retrieval. pp. 157–163. Paris, France (October 2002)
8. Berenzweig, A., Logan, B., Ellis, D.P.W., Whitman, B.: A large-scale evaluation of acoustic and subjective music similarity measures. In: Computer Music Journal (2003)
9. Chalmers, M.: A linear iteration time layout algorithm for visualising high-dimensional data. In: Proceedings of the 7th conference on Visualization '96. pp. 127–ff. VIS '96, IEEE Computer Society Press, Los Alamitos, CA, USA (1996), http://portal.acm.org/citation.cfm?id=244979.245035
10. Cooper, M., Foote, J., Pampalk, E., Tzanetakis, G.: Visualization in audio-based music information retrieval. Comput. Music J. 30, 42–62 (June 2006), http://portal.acm.org/citation.cfm?id=1176357.1176365
11. Dittenbach, M., Berger, H., Genswaider, R., Pesenhofer, A., Rauber, A., Lidy, T., Merkl, D.: Shaping 3d multimedia environments: the mediasquare. In: Proceedings of the 6th ACM international conference on Image and video retrieval. pp. 85–88. CIVR '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1282280.1282292
12. Ehringer, D.: The dalvik virtual machine architecture (2010)
13. Ellis, D.P.W., Whitman, B.: The quest for ground truth in musical artist similarity. In: in Proc. International Symposium on Music Information Retrieval ISMIR-2002. pp. 170–177 (2002)
14. Erten, C., Kobourov, S.G., Le, V., Navabi, A.: Simultaneous graph drawing: Layout algorithms and visualization schemes. In: In 11th Symposium on Graph Drawing (GD. pp. 437–449 (2003)
15. Frank, J., Lidy, T., Peiszer, E., Genswaider, R., Rauber, A.: Creating ambient music spaces in real and virtual worlds. Multimedia Tools Appl. 44(3), 449–468 (2009)
16. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw., Pract. Exper. 21(11), 1129–1164 (1991)
17. Gansner, E., North, S.: Improved force-directed layouts. In: Whitesides, S. (ed.) Graph Drawing, Lecture Notes in Computer Science, vol. 1547, pp. 364–373. Springer Berlin / Heidelberg (1998), http://dx.doi.org/10.1007/3-540-37623-2_28, 10.1007/3-540-37623-$2_2$8
18. Geleijnse, G., Korst, J.: Tagging artists using cooccurrences on the web. In: Proceedings Third Philips Symposium on Intelligent Algorithms (SOIA 2006), pages 171 – 182 (2006)
19. Hayashi, K., Inoue, M., Masuzawa, T., Fujiwara, H.: A layout adjustment problem for disjoint rectangles preserving orthogonal order. In: Proceedings of the 6th International Symposium on Graph Drawing. pp. 183–197. GD '98, Springer-Verlag, London, UK (1998), http://dl.acm.org/citation.cfm?id=647550.728930

20. Holten, D., van Wijk, J.J.: Force-directed edge bundling for graph visualization. Comput. Graph. Forum 28(3), 983–990 (2009)
21. Huang, X., Lai, W.: Force-transfer: A new approach to removing overlapping nodes in graph layout. In: IN GRAPH LAYOUT, IN '25TH AUSTRALIAN COMPUTER SCIENCE CONFERENCE', ADELAIDA. pp. 349–358 (2003)
22. Inc., G.: http://developer.android.com/reference/android/app/Activity.html
23. Kobourov, S.G.: Force-directed drawing algorithms (2004)
24. Li, W., Eades, P., Nikolov, N.: Using spring algorithms to remove node overlapping. In: proceedings of the 2005 Asia-Pacific symposium on Information visualisation - Volume 45. pp. 131–140. APVis '05, Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2005), http://dl.acm.org/citation.cfm?id=1082315.1082334
25. Lidy, T., Rauber, A.: Evaluation of feature extractors and psycho-acoustic transformations for music genre classification. In: Proceedings of the Sixth International Conference on Music Information Retrieval (ISMIR 2005). pp. 34–41. London, UK (September 11-15 2005)
26. Marshall, B.: Aggregating music recommendation web apis by artist. In: Information Reuse and Integration (IRI), 2010 IEEE International Conference on. pp. 75 –79 (2010)
27. Mayer, R., Neumayer, R., Rauber, A.: Rhyme and style features for musical genre classification by song lyrics. In: ISMIR. pp. 337–342 (2008)
28. McFee, B., Lanckriet, G.: Heterogeneous embedding for subjective artist similarity. In: Tenth International Symposium for Music Information Retrieval (ISMIR2009)) (October 2009)
29. McFee, B., Lanckriet, G.: Partial order embedding with multiple kernels. In: Proceedings of the 26th Annual International Conference on Machine Learning. pp. 721–728. ICML '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1553374.1553467
30. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. J. Vis. Lang. Comput. 6(2), 183–210 (1995), http://dblp.uni-trier.de/db/journals/vlc/vlc6.html#MisueELS95
31. Morrison, A., Ross, G., Chalmers, M.: Fast multidimensional scaling through sampling, springs and interpolation. Information Visualization 2, 68–77 (March 2003), http://dx.doi.org/10.1057/palgrave.ivs.9500040
32. Muelder, C., Provan, T., Ma, K.L.: Content based graph visualization of audio data for music library navigation. In: In Proceedings of The IEEE International Symposium on Multimedia (ISM2010) (December 2010)
33. Neumayer, R., Dittenbach, M., Rauber, A.: Playsom and pocketsomplayer: Alternative interfaces to large music collections. In: Proceedings of the Sixth International Conference on Music Information Retrieval (ISMIR 2005). pp. 618–623. London, UK (September 11-15 2005)
34. Noack, A.: An energy model for visual graph clustering (2003), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.3319
35. Pampalk, E., Rauber, A., Merkl, D.: Content-based organization and visualization of music archives. pp. 570–579. ACM (2002)
36. R. Tesoriero, J. A. Gallud, M.D.L., Penichet, V.M.R.: HCI Design Patterns for Mobile Applications Applied to Cultural Environments HCI Design Patterns for Mobile Applications Applied to Cultural Environments, vol. 1, chap. 14, pp. 262–264. InTech (October 2008)
37. Rauber, A., Pampalk, E., Merkl, D.: Using psycho-acoustic models and self-organizing maps to create a hierarchical structuring of music by musical styles. In: Proceedings of the 3rd International Symposium on Music Information Retrieval. pp. 71–80.

Paris, France (October 13-17 2002), http://www.ifs.tuwien.ac.at/ifs/research/publications.html
38. Sarmento, L., Gouyon, F., Costa, B.G., Oliveira, E.C.: Visualizing networks of music artists with rama. In: WEBIST. pp. 232–237 (2009)
39. Scheib, V., Engell-Nielsen, T., Lehtinen, S., Haines, E., Taylor, P.: The demo scene. In: ACM SIGGRAPH 2002 conference abstracts and applications. pp. 96–97. SIGGRAPH '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/1242073.1242125
40. Whitman, B., Lawrence, S.: Inferring descriptions and similarity for music from community metadata. In: In Proceedings of the 2002 International Computer Music Conference. pp. 591–598 (2002)

9 Appendix A