BROWSING MUSIC RECOMMENDATION NETWORKS

Klaus Seyerlehner
Dept. of Computational Perception
Johannes Kepler University
Linz, Austria
klaus.seyerlehner@jku.at

Peter Knees
Dept. of Computational Perception
Johannes Kepler University
Linz, Austria
peter.knees@jku.at

Dominik Schnitzer
Austrian Research Institute for AI
Vienna, Austria
dominik.schnitzer@jku.at

Gerhard Widmer
Austrian Research Institute for AI
Vienna, Austria
gerhard.widmer@jku.at

ABSTRACT

Many music portals offer the possibility to explore music collections via browsing automatically generated music recommendations. In this paper we argue that such music recommender systems can be transformed into an equivalent recommendation graph. We then analyze the recommendation graph of a real-world content-based music recommender systems to find out if users can really explore the underlying song database by following those recommendations. We find that some songs are not recommended at all and are consequently not reachable via browsing. We then take a first attempt to modify a recommendation network in such a way that the resulting network is better suited to explore the respective music space.

1. INTRODUCTION

Now that millions of songs are available for purchase and download on modern music platforms, developing concepts that help customers to navigate and explore the underlying song database becomes more and more important. A straight forward solution that is used in many commercial settings to assist users in finding songs in a database is to simply present lists of recommendations. Users are then able to explore a collection by moving from recommendation to recommendation. Exploring a music collection via such a sequence of recommendations is called *browsing*. We believe that browsing will be a key feature of modern music portals and consequently it is important to view recommendation not just in terms of individual recommendation queries only, but also as a continuous process. To analyze recommender systems with respect to their ability to support users to browse throughout a music collection, we can view a music recommender as a recommendation network. Recent research work on analyzing music recommendation networks [1, 2] indicates that many songs in such a network stay hidden in the so-called Long Tail [3,4]. One reason why songs stay hidden in the Long Tail is that it is hard to navigate through the network to reach those unknown songs. Thus, it seems to be an essential property of such a recommendation network that each song can be reached via browsing the recommendations. The goal of this paper is to analyze music recommendation networks with respect to their *browsability*.

The rest of this paper is organized as follows: In section 2 we start with formally defining the general recommendation scenario. In section 3 we show that under some restrictions any recommender system can be transformed into an equivalent *recommendation graph*. We then define properties for a recommendation graph that make such a graph useful for browsing the underlying music database and introduce the notion of a *browsing graph*. In section 4 an analysis of a recommendation graph of a real world content-based music recommender system illustrates the limitations of a simple recommender system with respect to the reachability of database items. We then propose in section 4.2 an algorithm which effectively modifies a recommendation graph to overcome these reachability limitations. Finally, we give an outlook on the application of the proposed method and some future work.

2. RECOMMENDATION SCENARIO

Although many different music recommender systems have been proposed so far, the fundamental principle is basically the same. Independent of the actual recommendation approach we can give a **formal model of a recommendation scenario** for item-based recommendation:

Given a set of database items U of size N and a specific item $o \in U$ that a user is currently focusing on, a *recommendation* is a subset of items $R \subset U$ related to o, where the size of the subset R is far smaller than the total number of items in the database. This very simple recommendation scenario can be extended by generating a recommendation not only based on the current item o but additionally spec-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.
© 2009 International Society for Music Information Retrieval.

ifying a user profile $p \in P$, where P is a set of all user profiles stored in the recommender system. We call a tuple $q = (o, p)$ a *recommendation query* and the item set $R(q)$ returned by the recommender system the *result set* or *recommendation*.

Actual recommender systems then differ in the way the recommendations are generated in this scenario. With respect to music recommender systems, there seem to exist five general recommendation approaches: collaborative filtering approaches, content-based approaches, web-mining based approaches, expert-based approaches and hybrid approaches.

Our investigations in the next sections are in general independent of the recommendation approach. The only requirement is that the recommender system under investigation returns, for any query $q(o, p)$, an ordered set of recommended items of a given length k such that the recommended items are ordered according to a measure of relatedness.

3. RECOMMENDATION GRAPHS

An intuitive way of exploring a music catalog is to pick an arbitrary item out of the database and than navigate throughout the database moving from recommendation to recommendation. One important requirement of such a browsing system is reachability. Reachability essentially ensures that a user will be able to access all songs in the collection by means of exploration and will not be limited to a small subset by the recommender system. To be able to show that reachability is ensured for a specific recommendation algorithm, we have to establish a formal model of the browsing process.

Based on our definition of a recommendation scenario (see section 2), *browsing* can be seen as an extension to recommendation from a single query to a consecutive sequence of queries $s = (q_1, q_{,2}, ..., q_N)$. Two consecutive queries $q_i = (o_i, p)$ and $q_{i+1} = (o_{i+1}, p)$ within such a browsing sequence are related by the fact that the item o_{i+1} of the next recommendation query q_{i+1} is an element of the result set of the previous recommendation query q_i. Consequently, a sequence s of recommendation queries of length N is a valid browsing sequence in the case that the following property is fulfilled:

$$\forall i < N : o_{i+1} \in R(q_i) \tag{1}$$

To guarantee this essential reachability property for a recommender system we have to show that starting from a arbitrary but fixed database item, all other database items can be reached by a finite sequence of recommendation queries. Formally, **reachability** starting from an arbitrary but fixed item o_1 holds if:

$$\forall o \in U : \exists i \in \mathbb{N} : \forall j < i : \\ o_{j+1} \in R(q_j) \wedge q_{j+1} = (o_{j+1}, p) \wedge o \in R(q_i) \tag{2}$$

Before we can start drawing any conclusions about reachability, we have to make some additional assumptions about the recommender system. The reason is that for dynamic recommenders, e.g., based on collaborative filtering, where the recommendations may change as a result of system use, it is impossible to prove reachability, since we cannot make any assumption about future recommendations. Therefore we have to assume a *static* recommender system where the recommendation will not change over time. It is important to note that this is not in principle a loss in generality; it just implies that if there are any changes in the recommender system then we also have to prove reachability for this new recommendation state.

Furthermore we constrain our analysis to systems where the recommendation result is independent of the user profile. This implies that all users get the same recommendations for one and the same query item. Once more this is not in principle a loss in generality as we could handle such systems by proving reachability for each user separately. In practice, however, analyzing recommender systems that generate personalized recommendations seems to be impossible due to the potential enormous computational costs.

Given these restrictions, we can now transform every possible recommender system into a *recommendation network* or *recommendation graph*. A recommendation graph is a directed graph $G = (V, E)$, where each vertex in the graph corresponds to a database item. For each item o in the database the corresponding vertex in the graph has a directed edge to all the items in the result set $R(q)$ of the recommendation query $q = (o, p)$. (Note that based on our assumptions $R(q)$ does not depend on p, an optionally given user profile.) To prove reachability for such a recommendation graph we can for instance apply the *depth first search* algorithm for each vertex in the graph separately.

While this is not a very practical or fast method to prove reachability, in most cases it is quite trivial to disprove reachability either by showing that the recommendation graph is not connected, or by identifying a single *source*. A source is a vertex v which has no incoming edges, i.e., has an indegree of zero ($\deg^-(v) = 0$). This implies that there is a song in the database that does not occur in the result set of any possible recommendation query and is consequently not reachable at all. Sources are especially problematic with respect to browsing: not only are they not reachable if one starts from some specific song in the database, but they are not reachable from *any* other song in the database. In section 4 we show, based on empirical analysis of a real world music recommender system, that in contrast to what one would expect it is rather likely that there are many sources in a simple recommendation graph. Identifying sources in a graph is a fast operation and can be done in $O(n)$.

Proving and disproving reachability is of course an important analysis, however in the likely case that we are able to disprove reachability, what can we do about it? How can we find a recommendation algorithm that guarantees reachability? To put it another way, can we modify a recommendation graph in such a way that the recommendation graph guarantees reachability? In section 4 we will

show that it is quite likely that a recommendation graph does not fulfill the reachability property. We then propose an algorithm that transforms a *recommendation graph* into a *browsing graph*, a recommendation graph that besides reachability has some other properties that we are going to introduce in the next section.

3.1 Further Requirements and Constraints

Up to now we have only considered reachability as an important property of a recommendation graph. But we can derive additional constraints for the recommendation graph by analyzing user requirements of browsing systems.

The first requirement that jumps to the eye is that the result set should be relatively small — first of all, because the display space for recommendations is in general limited on output devices, and secondly, because too large a result set would confuse the user and make for a very unfocused search. Thus it is a natural constraint that the size of the result set should not exceed a maximum number of recommendations $k_{\max}$. For the corresponding recommendation graph this implies that the outdegree of all vertices is less or equal to $k_{\max}$. We call this property **maximum outdegree property**.

$$\forall v \in V : \deg^+(v) \leq k_{\max} \tag{3}$$

The second constraint is that if item B is a recommendation for item A then item A should also be a recommendation of item B. This corresponds not only to humans' intuition that similarity relations are symmetric, but also allows to easily go back each recommendation step. The **symmetry property** as defined in (4) implies that the browsing graph is an undirected graph.

$$\begin{aligned} &\forall e_1 = (v_1, u_1) \in E : \\ &\exists e_2 = (v_2, u_2) \in E : \\ &\quad v_1 = u_2 \wedge u_1 = v_2 \end{aligned} \tag{4}$$

Finally, we extend our notion of reachability. Reachability just ensures that starting from an arbitrary vertex there is at least a single path to each other vertex. This could make it rather difficult to find this path. Therefore we require each vertex to have a minimum number of incoming edges. For the browsing graph this implies that each vertex has a minimum indegree $k_{\min}$ and means that each item is reachable by recommendations from at least $k_{\min}$ other items. This property is called **minimum indegree property**.

$$\forall v \in V : \deg^-(v) \geq k_{\min} \tag{5}$$

As a result from this requirement analysis we claim that a recommendation graph is better suited for browsing a music archive if these four properties are ensured. We then call such a graph no longer a *recommendation graph*, but a *browsing graph* instead.

In the next section we illustrate the limitations of a simple recommendation graph based on a real world content-based music recommender system and show that in most cases such a recommendation graph is not adequate for browsing. We then introduce a heuristic algorithm that can transform a recommendation graph into a browsing graph.

4. BROWSING GRAPHS

4.1 An Empirical Study

In this section we will show that properties like reachability are essential and cannot be neglected when designing a recommender or browsing system. To do so we analyze a real world content-based music recommender system attached to the music portal. The FM4 Soundpark [1] is an internet platform of the Austrian public radio station FM4. This internet platform allows artists to present their music free of any cost in the WWW. All interested parties can download this music free of any charge. At the moment this music collection contains about 10000 songs and is steadily growing. In our experiments we were allowed to use a subset of 7665 songs out of the whole collection.

The recommender system attached to the FM4 Soundpark music portal is based on a standard similarity measure for music audio files. Each song is modeled as a distribution of local spectral features, namely Mel Frequency Cepstrum Coefficients (MFCCs). MFCCs are a compact representation of the spectral envelope of a short audio frame and are one of the most widespread features used in the Music Information Retrieval (MIR) community. A single multivariate Gaussian distribution is used to model the distribution of MFCCs of a song. Recommendations can then be generated by comparing these distributions. This is commonly done by computing the Kullback-Leibler (KL) divergence [5] or relative entropy between the distributions of two songs. For more details on the feature extraction process and the generation of music recommendations we refer to [6–8]. Using the MIR system of the FM4 Soundpark we were able to generate lists of recommended songs of a given length k, ordered according to the similarity to the query song, exactly as required by our general scenario (see section 2).

Assuming a fixed sized result set of k recommendations for each query, we systematically created all recommendation graphs for $k = 1 \ldots 100$, where we denote k as the degree of the recommendation graph. For each of these graphs we computed the indegree for all vertices and counted the number of sources in each graph. Figure 1 shows that for small result sets the number of sources is extremely high. For example, in the recommendation graph of degree 5 there are 2661 sources, which implies that 34.72% of all the songs in the music collection are not reachable at all within this graph. By increasing the result set size the number of sources decreases, but even for a quite large result set of size 20 we still have approximately 1320 sources. Consequently still 17.22% of the songs in the collection cannot be reached. From figure 2 we can see how the number of sources scales with the collection size. To simulate different collection sizes songs were randomly removed from the collection. Figure 2 illustrates that the problem gets worse for increasing collection sizes. In fact the analysis of the recommendation graph that corresponds to the online version of the FM4 Soundpark — there are only three recommendations per song — revealed

[1] http://fm4.orf.at/soundpark/main

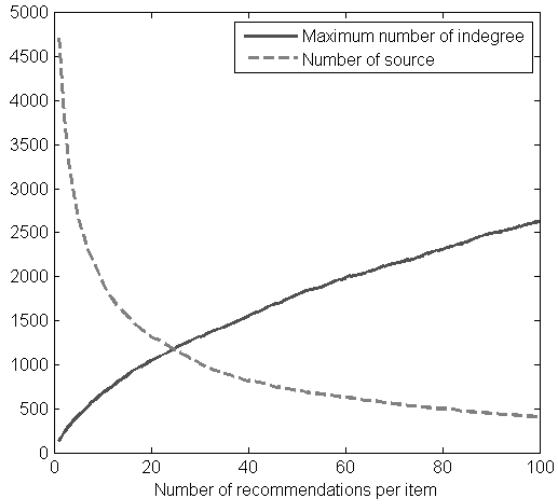

Figure 1. *For small result sets, the number of sources is extremely large and decreases with an increasing number of recommendations per query, whereas the maximum indegree over all vertices in each graph increases. For a result set size of 100, there is one song that appears in the recommendation list of 2628 other songs, or in 34.29% of all recommendation lists.*

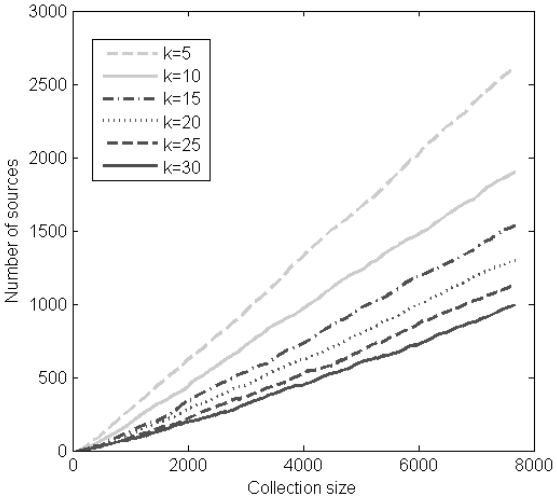

Figure 2. *The number of sources in a recommendation graph scales with the number of items in a database. Furthermore the number of sources depends on the number of recommendations for each query. This is illustrated for fixed result set sizes of* $k = 5, 10, 15, 20, 25, 30$.

that only 56,79% of all songs are reachable by recommendations, the remaining 43,21% of the songs are sources and are never recommended.

In addition to the number of sources, we also computed the maximum indegree over all vertices in each graph, visible in figure 1. Obviously, while some songs are not reachable at all, some others are directly reachable from very many songs. However it is of course quite implausible that a single song is similar to several hundred other songs. Songs that have a very high indegree, but do not share any perceptual similarity with the referring songs are called *hub*-songs according to [9]. In our case the hub problem seems to be related to the content-based audio similarity measure itself. Interestingly, hubs naturally appear in social networks (including collaboration networks) as well [10]. Regardless of the reasons for hubs and sources, both essentially reduce the usability of music recommender systems to explore the music spaces. In the following we propose a heuristic algorithm that transforms a recommendation graph into a browsing graph that fulfills the properties introduced in section 3.

4.2 Constructing a Browsing Graph

The main idea behind our approach is to transform a recommendation graph into a browsing graph, simply by replacing all directed edges by undirected edges and then iteratively (and heuristically) removing edges from the resulting graph such that the *maximum outdegree* and the *minimum indegree* property are satisfied for all vertices. The symmetry property is automatically ensured because the graph is undirected. Furthermore, reachability is guaranteed if the resulting graph is connected.

The proposed algorithm has three important parameters. There is the minimum indegree $k_{\min}$ and the maximum outdegree $k_{\max}$, which directly result from the required properties. It is easy to see that in combination with the symmetry property this implies that each vertex in the final browsing graph will have to have an edge degree between $k_{\min}$ and $k_{\max}$. The proposed algorithm starts from the directed version of the recommendation graph. One could of course start the algorithm from a recommendation graph with outdegree $k_{\max}$, but since we want to give our algorithm additional flexibility during the process of removing edges, it is required that the original recommendation graph has an outdegree of at least k_{start} for all vertices. This simply means that for each item we can generate at least k_{start} recommendations and is in line with the requirement on recommender systems in section 2. The three parameters are related to each other as stated in (6).

$$k_{\min} < k_{\max} < k_{\text{start}} \tag{6}$$

The only thing left to do is to remove edges till each vertex has a degree in between $k_{\min}$ and $k_{\max}$. This should be done in such a way that each vertex tries to remove its 'weakest' links (i.e., those with the lowest degree of relatedness), since the recommendations should be as good as possible. This can be done as follows:

1. Put all vertices into a priority queue q, where all vertices are sorted according to their degree $\deg(v)$; break ties among same-degree nodes randomly;
2. Pop the vertex with the highest degree from the queue.
3. If this vertex already has a degree smaller than or

equal to $k_{\max}$, then all vertices in the queue have a degree smaller or equal to $k_{\max}$. We are done.

4. As the current vertex has too many edges, remove an edge that connects this vertex to another vertex having a degree greater than $k_{\min}$. Choose the edge to remove according to the indegree of the neighboring vertices. Remove the edge connecting to the vertex with the highest indegree and if there are several vertices of the same indegree remove the vertex with the weakest (lowest similarity) edge. If this vertex is not connected to any other vertex having a degree greater than $k_{\min}$, then we are not able to ensure the maximum indegree property for this node. Stop in this case.

5. Since we have removed an edge, the indegrees of the two vertices connected by the edge have changed. Remove them from the queue and reinsert them such that the queue is up to date.

6. Go back to step 2.

Of course it is true that this algorithm might find a solution where individual vertices have an edge degree higher than $k_{\max}$, violating the maximum outdegree property. This can be due to the fact that for given constraints there simply does not exist any solution. In such a case weakening the constraint till enough solutions to the problem exist can help. If there are enough solutions, simply rerunning the algorithm might help. Vertices of the same edge count are inserted into the priority queue in random order. Therefore the algorithm might find other solutions. However our experiments indicate that it is quite easy to find a valid solution. Furthermore, the proposed algorithm does not guarantee that the resulting graph is connected, but in all our conducted experiments the resulting browsing graph turned out to be connected.

4.2.1 Time Complexity

One major advantage of this algorithm is that it is of time complexity $O(n \log(n))$. At most $n(k_{\text{start}} - k_{\min})$ edges have to be removed. Therefore we have to perform a maximum of $3n(k_{\text{start}} - k_{min})$ removal or insertion operations on the sorted priority queue. Sorting and removing elements from a priority queue can be done in $O(\log(n))$, e.g., by using a balanced red-black tree. Therefore removing all the additional edges from the graph can be done in $O(n \log(n))$. The initial insertion operation of all elements in the priority queue is also of complexity $O(n \log(n))$. Thus, the overall complexity of this algorithm is $O(n \log(n))$.

4.2.2 Validation of the Transformation Algorithm

To validate the proposed algorithm we analyzed the result after the transformation of the FM4 Soundpark into a browsing graph. The parameters used to transform the graph were $k_{\min} = 4$, $k_{\max} = 7$ and $k_{\text{start}} = 9$. As we do not have yet statistics of the usage before and after the transformation, we follow the standard procedure in MIR research and evaluate transformation algorithm in an indirect way, via a music genre analysis. For all query songs q we count the number of songs in the result set $R(q)$ that have the same genre as the query song and compute the overall percentage relative to the number of recommended songs. That way we measure the accuracy of the recommendations independent of the number of the recommendations. The accuracy of the recommendations using result sets of length $k = 5$ was 35.39%, for $k = 6$ was 34.86% and for $k = 7$ was 34.32%. After the transformation using the above parameters the accuracy was 35.63% with an average degree of 5.918 per vertex. This preliminary result indicates that there is only a marginal change in recommendation quality, however a more detailed empirical study will be done in future. Furthermore to evaluate how

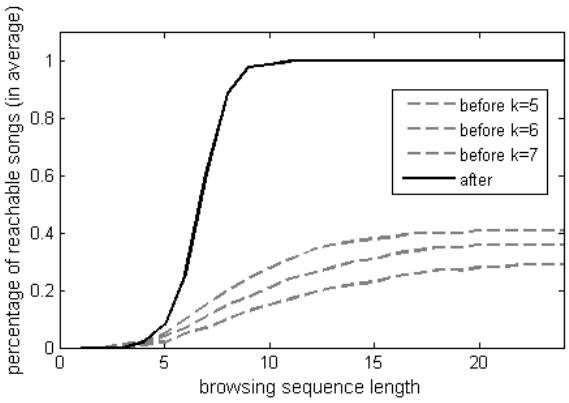

Figure 3. *The average percentage of songs that can be reached by browsing sequences of different length. Before the transformation (for $k = 5, 6, 7$) and after the tranformation.*

the reachability of songs has changed we investigated how many songs can be reached in average by a recommendation sequence of length l. To do so we computed for each song the number of songs that can be reached by such a sequence. This can be done by traversing the recommendation graph using the *breadth-first search (BFS)* algorithm up to a maximum depth of l. We then take the average over all songs to get a quality indicator for the whole network. As one can see from figure 3 after the transformation more songs can be reached when browsing the resulting graph than before.

5. APPLICATION AND FUTURE WORK

Based on the graph-theoretic studies performed on the FM4 Soundpark Recommender, we are now investigating ways of turning the Soundpark into a Browsing Graph. Given the purpose of the system – to make new music artists known to a wide public – reachability of as many artists (or works) as possible would be a prime feature. This is not quite straightforward and will involve some interesting research questions. Several aspects have to be addressed:

Recommendation quality: Clearly, the quality of the recommendations changes as a recommendation graph

(which is based on content-based similarity relations) is transformed into a browsing graph (which sacrifices certain recommendation links in order to satisfy the browsing constraints). Whether or not that unduly degrades the quality of the recommendation service can only be studied empirically. We will address this issue by means of a large-scale user study, which is yet to be designed (see below).

Incremental updates: The FM4 music database grows on a daily basis. Every day, dozens of new songs, mostly by new artists, are added to the database and integrated into the recommender system in nightly batch update sessions. Thus, the browsing graph transformation will also have to be run at regular intervals. As an alternative, we will look into the possibility of incremental update algorithms for browsing graphs.

Time-varying recommendations: A specific aspect of the growing database is that the system's recommendations may change from day to day. That is, if the user selects the same seed song on two consecutive days, she may get different recommendations of songs that are supposedly 'similar'. This may be a problem in certain applications, but perhaps not in the case of the Soundpark. Soundpark users have been taught to regard the recommendation service as a means to explore the Soundpark and find new things that they would not otherwise find. From the user feedback we currently have, we can conclude that many of the users are quite open-minded about occasional 'strange' recommendations, regarding them as 'interesting' or 'funny' ideas by the computer, rather than annoying mistakes. Thus, they might find time-varying recommendations (if they ever notice them) to be enriching rather than irritating.

Modifications to the Soundpark recommender system will be accompanied with a large scale *user study*. We have access to two kinds of user feedback: the browsing sessions themselves (click data) as logged by the Soundpark server, and an on-line user forum, where users discuss their impressions of the system (among other things). Questions to be studied include, e.g., whether improved reachability conditions really increase the number of artists that are listened to by users; whether and how one can quantify differences in recommendation quality between recommendation and browsing graphs; and general aspects of user browsing behaviour that may help in designing better recommenders in the future (for instance: how long is a typical browsing sequence? do users follow more than one recommendation in a given recommendation list? etc.).

In this way, the FM4 Soundpark may then become one of the first real-world music recommendation system that is (a) purely content-based, that is, based on musical similarity as estimated by the system itself, and (b) specifically designed to maximize the percentage of music items that can be found via similarity-based browsing.

6. CONCLUSIONS

In this paper we have shown that designing music recommender systems is not as straight forward as it seems. Especially reachability is an important property if a music recommendation system should also allow users to explore a music archive via browsing. A bad system design might have the consequence that a portion of all songs in the database cannot be discovered as they are not accessible at all. To overcome these limitations we took a first attempt to modify the graph representation of a recommender system in such a way that browsing the resulting recommendation network is more convenient. We believe that improving the accessibility of songs in a music archives can significantly increase the usability of music services and might even help to alleviate the long tail phenomenon by ensuring the accessibility of *'niche'* products.

7. ACKNOWLEDGEMENTS

This research was supported by the Austrian Research Fund (FWF) under grant L511-N15, and by the Austrian Research Promotion Agency (FFG) - project number 815474.

8. REFERENCES

[1] O. Celma and P. Cano. From hits to niches? or how popular artists can bias music recommendation and discovery. In *2nd Workshop on Large-Scale Recommender Systems (ACM KDD)*, Las Vegas, USA, 2008.

[2] O. Celma and P. Herrera. A new approach to evaluating novel recommendations. In *2008 ACM Conference on Recommender Systems*, Lausanne, Switzerland, 2008.

[3] C. Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, July 2006.

[4] E. Brynjolfsson, Y. J. Hu, and M.l D. Smith. From niches to riches: Anatomy of the long tail. *Sloan Management Review*, 47(4), 2006.

[5] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 1951.

[6] M. Levy and M. Sandler. Lightweight measures for timbral similarity of musical audio. In *Proc. of the 1st ACM Workshop on Audio and Music Computing Multimedia*, Santa Barbara, USA, 2006.

[7] M. Mandel and D. Ellis. Song-level features and svms for music classification. In *Proc. of the 6th Int. Conf. on Music Information Retrieval*, September 2005.

[8] A. Flexer, D. Schnitzer, M. Gasser, and G. Widmer. Playlist generation using start and end songs. In *Proc. Int. Sym. on Music Information Retrieval*, 2008.

[9] J.-J. Aucouturier and F. Pachet. A scale-free distribution of false positives for a large class of audio similarity measures. *Pattern Recogn.*, 41(1):272–284, 2008.

[10] A.-L. Barabsi R. Albert. Statistical mechanics of complex networks. *Review of Modern Physics*, 2002.