Song Clustering Using Peer-to-Peer Co-occurrences

Yuval Shavitt
School of Electrical Engineering
Tel-Aviv University, Israel
shavitt@eng.tau.ac.il

Udi Weinsberg
School of Electrical Engineering
Tel-Aviv University, Israel
udiw@eng.tau.ac.il

***Abstract*—Peer-to-peer (p2p) content sharing networks are commonly used by millions of users for sharing music files, often performed by artists even before becoming mainstream. In such networks, as well as modern web 2.0 services, users with similar musical taste often share similar files. This results in songs that have similar properties to be shared together by many users, where the higher the number of song co-occurrences in different users, the stronger is the indication of a tight relationship between these songs. In this work we leverage this feature and propose methods for detecting these "natural" clusters of similar songs. The resulting clusters are shown to be useful in recommender systems, as they almost mitigate the need to use meta-data which is known to be noisy due to its user-generated nature.**

We present data collected from the Gnutella network and its properties and show two techniques for recommending content to users, one is based on clustering similar-minded users and the other creates song similarity graph and maps users to clusters based on their songs. We show that both techniques result in relatively accurate recommendations, indicating that p2p networks can be leveraged for creating useful recommender systems that can be used for easier content retrieval.

I. Introduction

Peer-to-Peer (p2p) networks are commonly used by millions of users for sharing content. Regardless of the legal aspects, music files (e.g., mp3 files) and even complete albums are often published by users in the p2p network, so that other users that share similar taste in music can download the files to their local storage. Moreover, it has been shown [1] that these networks can help spot emerging trends before they become mainstream.

However, searching for files in p2p networks can often be quite frustrating. A recent study [2] showed that only 7-10% of the queries in the Gnutella [3] network are successful in returning useful content. Therefore, building a recommendation system that takes advantage of the vast amount of data that exists in p2p networks and its properties, while operating efficiently, is beneficial.

Recommender systems were suggested to help users find new content based on their preferences or similarity to other users. These systems have been studied extensively in recent years (for an extensive study on music recommender systems see [4]), mostly relying on the willingness of users to rank their preferences in order to provide better recommendation. However, p2p networks, alongside with new home-entertainment services such as IPTV, introduce new opportunities and challenges to standard recommender systems, making them somewhat unfitted for this task.

First, in p2p networks, users do not explicitly rank their preferences, but simply download content, use it if they like it or delete (or simply ignore) it if they do not like what was downloaded. This *implicit ranking* makes it difficult to assess whether a user "likes" or "dislikes" the downloaded content. Notice that using listening habits, which is also considered as implicit ranking [4], provides more information about whether and how much a user likes a song than exists in p2p network. Additionally, in every user generated content, there is a large amount of "noise". This results in an abundance of duplicate content with different titles, multiple and even conflicting tagging and spelling mistakes or ambiguities. Finally, the data is extremely sparse since there are lots of users sharing lots of content, but a given user holds only a tiny fraction of the content. Unlike systems where explicit ranking exists, allowing any user to rank any content, a user in p2p networks only "ranks" content that was explicitly downloaded.

On the other hand, due to their community based nature, music files in p2p networks and in modern web 2.0 services are very dynamic and quickly follow trends [1], [5]. As such, p2p networks, which are the focus of this paper, have the potential to uncover hypes and active trends in music as these networks cover a wide range of audience. Many social networks are somewhat biased towards their target audience. For example, we compared the top artists in the United States in *last.fm* [1] and the BillBoard [2] Hot 100. Only a few artists were found in both, whereas almost all of BillBoard's top artists were found among the top popular artists in our p2p dataset.

In this work, we use the observation that in p2p networks, as well as web 2.0 services, users with similar musical taste often share similar files. This results in songs that have similar properties to be shared together by many users,

[†]This research was supported in part by a grant from the Israel Science Foundation (ISF) center of excellence program (grant number 8008/03); and by the Next Generation Video (NeGeV) consortium, Israel.

[1]http://ws.audioscrobbler.com/
[2]http://www.billboard.com/

where the higher the number of song co-occurrences in different users, the stronger is the indication of a tight relationship between these songs. We leverage this feature and propose methods for detecting these "natural" clusters of similar songs, allowing us to overcome the lack of explicit ranking, the noisy meta-data and the extreme dimensions and sparseness of the network.

The novelties presented in this paper are threefold. First, we present the data collected from the Gnutella p2p network and detail its inherent complexities and the techniques used for cleaning and processing it. We then present a method that reduces the sparsity and works without explicit ranking, by quantifying how much a user likes each artist, and show it is possible to use this data for efficient artist recommendation. This however is achieved using the meta-data attached to each song. Finally, we show a method for using the suggested global clustering for overcoming all of the discussed difficulties and creating a song recommender system. The techniques presented in this paper have the potential to be either implemented as centralized algorithms in web 2.0 services or be distributed in p2p networks, allowing users to easily find content based on their dynamically changing preferences or shared storage.

II. Peer-to-Peer Dataset

The data used in this paper was collected from the music files that are shared in the Gnutella [3] p2p network. The files were collected by crawling for 24 hours over the shared folders of over 1.2 million users during November 25th 2007, selecting only files that correspond to musical content (.mp3 files). The crawl did not download the content of the files but only their unique identifiers and meta-data which is required for the evaluation of the algorithms. At the time of the crawl, Gnutella was the most popular file sharing network [6].

Since the data in the p2p network is mostly user generated, it is required to reduce the "noise" before processing it. One such noise is expressed in the form of spelling mistakes in the titles and tags and slightly different versions of the same song. To reduce this type of noise, we merged occurrences of the same song with different titles by ordering all the titles in a lexicographical order and then compared the edit distance [7] between consecutive titles in the sort order. In cases where the edit distance was smaller than 3 we merged the two files. This preliminary filter is used to reduce duplicates. Additional filters that are used to reduce invalid content and spelling mistakes in the tags themselves are described in the following sections.

After the initial filtering, we created a 2-mode graph, having users and song files (over 530k song files) as vertices. A link is created between a user and a song when the user shares the song.

Validating the results of the analysis is achieved using a smaller dataset created by randomly sampling a set of 100k users. The number of songs in the sampled set of users is 511k, a value which is not much lower than the 530k songs in the original crawl using 1.2 million users. This shows that most users in the p2p network share similar files and suggests that it is not needed to perform an exhaustive crawl in order to obtain sufficient representative data. The songs that are lost during the sampling belong to small and very specific musical niches, which are of less interest to this work since this music is usually known to the users that like it and they do not need a recommendation system in order to find it.

Further examining the distribution of the number of songs that each user shares shows that it starts with a power-law regime which is cut-off around 150 songs per user. We attribute this to the finite amount of disk space users are willing to devote for sharing or to the actual amount of different songs that are of interest to a user.

III. Artist Recommendation

The first usage of the data is for recommending less-popular artists to users. Formally stating the problem: given a user with known songs for the N most popular artists, recommend the user with R less-popular artists that she will like with high probability. This problem becomes extremely stressed when using data from p2p networks since less popular content is much harder to find than popular content [2]. Note that this method is not well suited for non-main-stream users, since they will hardly have any top-popular artists and therefore cannot benefit from this technique.

In order to address this problem, it is first required to quantify how much each user "likes" each artist. For this end, a user-to-artists matrix was created by counting, for each user, the number of songs of the 150 most popular artists (i.e., artists that appear the most times in the complete dataset). Assuming that $S(i, j)$ is the matrix that holds the number of songs of artist j that user i shares, we normalize it to reflect how much a user likes a given artist in the range of 0 to R as shown in Eq. 1. The value 0 means that a user does not share any song of the specific artist and we selected $R = 5$ that means that all of the user's shared songs are of the specific artist.

$$\hat{S}(i,j) = R \cdot \frac{|S(user_i, artist_j)|}{|\sum_{k=1}^{|artists|} S(user_i, artist_k)|} \qquad (1)$$

Notice that the creation of this matrix requires using meta-data in the mp3 files. Although it might be possible to obtain accurate meta-data for various web 2.0 or other controlled systems, p2p networks have very noisy meta-data. Overcoming this difficulty is achieved using the technique described in Sec. IV.

Initially, a C4.5 decision tree [8] was applied on $\hat{S}$ to find a predictive model for the 101st artist given the first 100 more popular artists. In the dataset the 101st artist is the band "Stained". Validation was done using a 10-fold cross validation. Despite the impressive 92% overall correctly classified success rate, the model mostly suggested $\hat{S}$ = "0" (i.e., "do not like") with 93% success, while $\hat{S}$ $=''$ 1" got 46% success and $\hat{S}$ = "2" in only 7% correctly classified. Even worse, the recall (i.e., how many were correctly classified out of the real data) is only 15% for $\hat{S}$ = "1" and less than 2% for $\hat{S}$ = "2". This stresses the severity of the sparseness problem that exists in the dataset, even after the dimensionality reduction achieved by the transformation of user-to-song to user-to-artist matrices.

A. *Clustering Users by Artist Preference*

Using direct clustering on $\hat{S}$ values, we find clusters of users based on the Euclidean distance using all artists. The clusters were built using the classical k-means [9] algorithm with a training set of 60% of the users for which data for all 150 artists is known, and evaluation was done using the remaining 40% of the users for which only the top N artists are known. After the k clusters are created using the training set, each user in the evaluation set is assigned to the cluster that has a centroid which is the nearest (Euclidean distance using top N artists) among all clusters.

For each user in the evaluation set, we wish to recommend 10 less-popular artists and estimate how much she will like each of them. The recommendation is constructed in two methods – the first is by copying data from the centroid and the second is from the nearest neighbor within the same cluster (using top N artists). Theoretically, finding a nearest neighbor could be implemented without clustering. However, assuming that the users are distributed evenly among clusters, which should be the case given that k-means converges to a local minima, finding a nearest neighbor within the cluster is roughly k-times simpler than finding it in the complete data set.

In order to evaluate the correctness of the recommendations, the recommended artists set were sorted in descending order (from "like most" to "like least") and compared against the sorted set of the true data. Comparing these two vectors, we looked at the number of artists that exist in both vectors (*hit*), number of recommended artists that appear in the surrounding position (up to one up or down) of the true data set (*area*) and the root mean square error (*RMSE*) of the distance between the two vectors.

Evaluation was performed using several different values for k (the number of clusters), N (top artists) and recommendation method (centroid or nearest neighbor). We found that while k has no significant effect on the resulting success measures, higher values of N greatly improved them. Using recommendation from the nearest neighbor, k=50, 200, 400 and N=50 resulted in an average hit of 4.5, area of 2.2 and RMSE of 2.6. When using N=100, the average hit has raised to 6.9, area to 4.2 and RMSE down to 1.5. Additionally, when using N=50, there was almost no difference in the measures between using the centroid and using nearest neighbor, while with N=100 there was almost 50% degradation in most measures when using the centroid.

This analysis shows the importance of having sufficient data about user preferences. However, while the results show that it is possible to create a fairly good artists recommendation, this technique has several drawbacks. First, as mentioned above, extracting the name of the performing artist from the song is not always possible, since mp3 files often have missing or incorrect meta-data. Second, musical taste is usually more complex than simply liking an artist, since artists often have songs that fit different audience and one might like a song of a certain artist but hate others. Finally, this technique uses a set of popular artists under the assumption that most users have some of their songs, therefore it misses important data about many other songs that a user likes. These drawbacks drive us to develop a new method, which leverage data about all of the songs that users share, with minimal usage of meta-data.

IV. "Natural" Song Clusters

Following the above, we wish to classify users in a way that better captures their taste in music, in better granularity than artists or genres, while barely relying on the correctness, or even existence of meta-data. To achieve this, we make an assumption on the way people use p2p networks for sharing music files: two songs that are shared together by many users, have similar features, causing people to like both of them. Following this, each song has a set of songs that are co-shared with it by many users. These groups of songs can be clustered, where each cluster contains a set of songs that share similar musical features or are at least liked by users that have similar preferences. It is then possible to use these clusters for overcoming missing meta-data or, as we show, create a recommendation system that helps users find songs. This technique does not rely on explicit ranking or existence of meta-data. Moreover, it reduces the dimensionality and sparseness of the original network without significant loss of data.

A. *Building Song Clusters*

Clustering the songs is achieved by converting the original 2-mode users-to-songs graph into a 1-mode song similarity graph. The weight of a link connecting two songs is the total number of users that share both files. Using the evaluation dataset, this graph has a total of 531k different song files with more than 1 billion interconnecting links. The songs are indexed by descending order of their popularity, i.e., for $i < j$, song i is more popular (shared by more users) than song j.

Many of the links represent very weak relations between songs. These links usually correspond to invalid or scarce content. Since this data creates "noise" which can cause incorrect clustering, several levels of filters are employed.

During the collection of songs we only include links that appear in at least 16 different users. This guarantees that “weak” ties between songs do not appear in the similarity graph. The second filter keeps, for each file, only the top 40% links (ordered by descending similarity value) and not less than 10. After these preliminary filters, roughly 20 million undirected links remain, a number which still requires a careful and efficient design of the analysis process.

B. Similarity Graph Properties

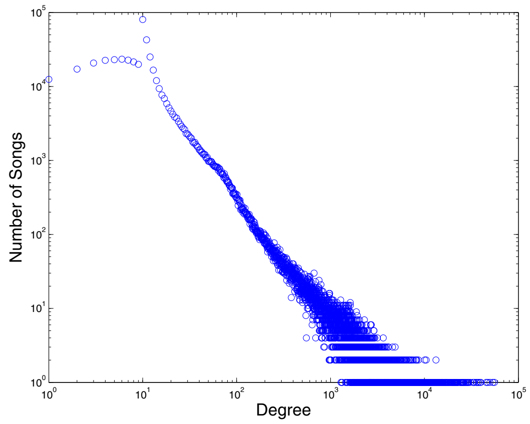


Figure 1: Songs degree distribution in the filtered similarity graph exhibiting power-law with broad set of degrees

The degree distribution of the similarity graph is given in Fig. 1, exhibiting a distinct power-law behavior with a broad range of degrees. The distortion seen in low degrees is attributed to the filtering. The first filter significantly reduces the number of low degree nodes, while the second filter increases the number of nodes that have a degree of 10. This power-law distribution suggests that there are relatively few songs with very high connectivity and many songs with low connectivity (a clear result of a preferential attachment process [10]).

Before any further processing, the weights of the links are converted from similarity values (high values indicate similar items) to distances (small values indicate similar items), as depicted in Eq. 2. In order to overcome the popularity bias [11] which is known to appear in collaborative filtering techniques, each link weight w_{ij} is normalized using a modified cosine-distance function of the popularity of both songs, C_i and C_j, and $-\log_2$ is applied on the result.

$$\hat{w}_{ij} = -\log_2\left(\frac{w_{ij}}{\sqrt{C_i \cdot C_j}}\right) \qquad (2)$$

For every two connected songs, the normalized distance represents the similarity between the two songs, while enabling to compare between distances regardless of the individual popularity of each song.

Since mostly strong links are useful for recommendations, we create smaller sub-networks that contain, for each song, only the top N neighbors, ordered by non increasing distance. This extends the basic filters since it uses the distance values, allowing it to capture the relative popularity of adjacent songs as well as the strength of their similarity. To make sure that these sub-networks do not significantly bias the results, we calculate the number of times each song appears as the nearest neighbor for N=1,5,10,20,50. The resulting distributions are very similar and for $N \geq 10$ the distributions almost overlap, showing that it is possible to use the smaller and denser sub-networks, while keeping a minimal bias in the results. Therefore, in this work we mostly use N=20 and N=50 for evaluation and name the graphs $TR20$ and $TR50$, respectively.

C. Clustering Algorithm

The clustering algorithm is a modified version of the well known k-means algorithm, so that it is fitted to handle large and sparse graphs. Our approach is to apply the clustering algorithm directly on the similarity graph. In this case, it is not possible to place arbitrary points in space (usually named “centroids”) and calculate the distance from them to all other vertices. Instead, only distances between songs in the graph can be calculated.

The algorithm first iteratively selects a set O of k songs that are used as the cluster origins. The origins are selected such that the distance between them is no less than a preset minimal distance. This ensures that clusters will not overlap, resulting in possible mixture of their features. Although it is possible to select cluster origins using slower but more sophisticated methods [12], the vast size of the network makes the random selection with minimal distance constraint very effective.

Distance between songs is calculated using a single-source Dijkstra [13] on the normalized distance along the shortest path. For each cluster origin, its distances from all other songs are calculated. Then, the algorithm scans all the songs in the graph and assigns each to the cluster whose origin is the nearest.

D. Properties of Song Clusters

For the clusters to be efficiently used by recommender systems, it is important that each cluster will have a large diversity in the popularity of its songs. This allows a recommender system to recommend songs that are less popular but are somehow related to popular songs that can be more easily associated with the preferences of a user.

Fig. 2 provides the distribution of song popularity in each cluster, using k=100 and $TR20$. For each cluster a box is plotted with a central mark being the median, the edges are the 25th and 75th percentiles and the whiskers extend to 1.5 times the difference between the 25th and the 75th percentiles. The figure shows that there is a large variety

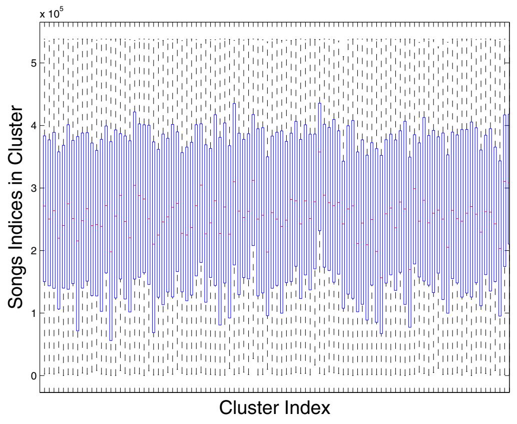


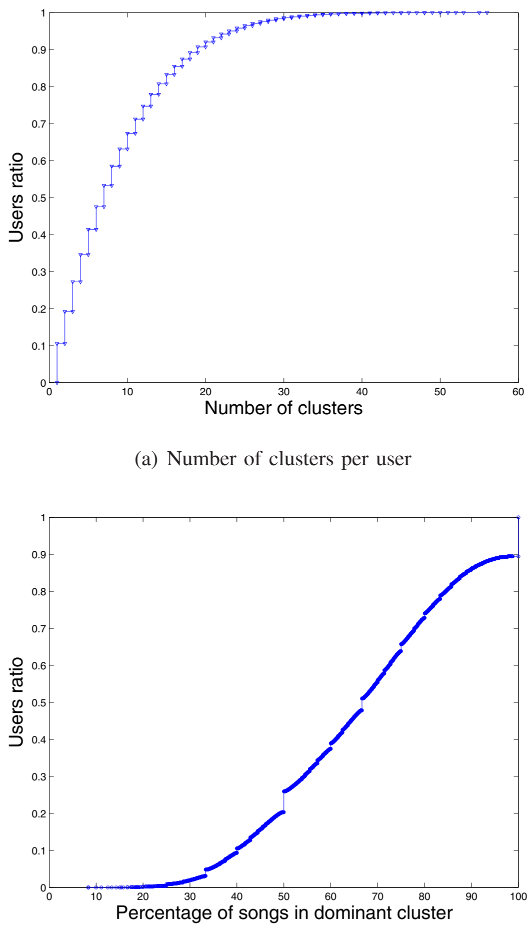
Figure 2: Distribution of songs popularity (low song index marks high popularity) using k=100, TR20, showing that clusters contain a mixture of popular and less-popular songs

of song indices in each cluster, i.e., each cluster holds a mixture of popular and less-popular songs. Furthermore, we found that most clusters have at least one song from the top 2,500 popular songs, and all of them have at least one from the top 5,000. We attribute this diversity to the observation that many users have diverse taste in music, and tend to like popular, as well as less-popular songs. As mentioned above, this property is especially useful for recommender systems since common collaborative filtering techniques are based on like-minded users causing them to support popular taste stronger than unpopular.

E. Mapping Users to Clusters

Since people often have a defined taste in music, it is expected that each user songs will be non-uniformly distributed over a small set of clusters, such that a large fraction of the songs reside in one cluster (the *dominant* cluster). This provides the ability to identify the musical taste of a user and provide more accurate recommendations.

Using the original user-to-song graph and the resulting song-to-cluster mappings, we count, for each user, the number of songs in each cluster. Fig. 3(a) plots the cumulative distribution function (CDF) of the number of clusters that each user has songs in. The figure shows that 11% of the users have all their songs in a single cluster and almost 70% are mapped to less than 10 different clusters. Fig. 3(b) plots the CDF of the percentage of songs (prevalence) each user has in the corresponding dominant cluster, showing a median value of almost 70%. The steep jump in the 100th percentile corresponds to the users that have all of their songs in a single cluster. Interestingly, evaluation using a different number of clusters (k=20,50) and network size (TR50, TR100) produced roughly the same results, showing that the cluster overlay consists of coherent clusters for a wide range of input parameters.



(a) Number of clusters per user

(b) Prevalence of dominant cluster

Figure 3: Empirical cumulative distribution function, resulting from mapping users to clusters using k=100, TR20, showing that (a) most users have songs in a few clusters, and (b) most dominant clusters has high prevalence

F. Recommending Songs

Creating recommendations for a user i is achieved by finding the dominant cluster of the user using her known shared songs. The dominant cluster of user i is a connected sub-graph D_i of the overall similarity graph. The recommendation algorithm builds the set of songs U_i that are shared by the user and are included in D_i. From the set of the remaining songs in the dominant cluster $V \in D_i \setminus U_i$ that are directly connected (i.e., have a direct link) to at least one song in U_i, the algorithm selects the song $v \in V$ that has the lowest average distance to all the songs in U_i. The algorithm recommends the song v and adds it to the set U_i. This selection process is repeated until no more recommendations are needed or until all songs in D_i exist in U_i.

Evaluating the correctness of the results is done by randomly selecting 30% of the songs of each user (training set). Then, the system attempts to recommend the songs in the evaluation set. We use two approaches: (i) recommend exactly 70% of the songs the user has, resolve the artists and see how many of the artists in the evaluation set appear in the recommendation (we use the SoundEx [14] algorithm when comparing artists to overcome spelling mistakes and ambiguities in user generated tags), (ii) find how far are the songs in the evaluation set from the training set, by dividing the distance of each song by the distance of the farthest song in the cluster minus the distance of the nearest song in the cluster. This serves as an indication to the "distance" the algorithm needs to traverse in order to recommend the songs in the evaluation set.

Evaluation using TR20 and k=100 results in a median precision of 12.1% and recall of 12.7%. The median average distance ratio (average is taken over all songs in each user's evaluation set) is 0.39. These are good results considering the amount of songs that exist and shared by p2p users. For example, one user had songs tagged with "Bob Dylan ft. Van Morrison", "Chuck Berry" and "Bob Dylan". Given only one song which was tagged with "Bob Dylan ft. Van Morrison", our algorithm recommended songs tagged with "Van Morrison". Although this is clearly a good recommendation, this recommendation was not counted as a "hit", since none of the recommended songs matched the evaluation set. This shows that it is possible yet not trivial to recommend songs and artists that a user may like. However, devising automatic criteria for an accurate estimation of the correctness of recommendations is complex.

Unlike collaborative-based recommendations systems (as the one presented in Sec. III-A), that find like-minded users for providing recommendations, the technique presented here maps a user to a cluster constructed from information that is spread over *all* users in the network. Capturing this global information can lead to much more accurate results than standard approaches.

V. Conclusion

This paper presents how songs sharing networks can be clustered based on co-occurrences of songs in shared lists of many users. Two methods for creating music recommender systems using data from p2p networks are presented. The first performs user clustering based on artist preferences, overcoming some inherent difficulties, namely implicit ranking, sparseness and extreme dimensions, while extensively relying on the existence of mp3 meta-data. The second extracts "natural" song clusters with minimal usage of meta-data and effectively manages to recommend songs to users. Evaluation is done using data obtained from browsing over 1.2 million Gnutella users. We show that both techniques result in successful recommendations based on very little known data. Considering that p2p networks, web 2.0 services and even IPTV share similar difficulties, all of these systems can benefit from the techniques presented in this paper and use them for improving user satisfaction.

References

[1] N. Koenigstein, Y. Shavitt, and T. Tankel, "Spotting out emerging artists using geo-aware analysis of P2P query strings," in *KDD '08*. ACM, 2008, pp. 937–945.

[2] M. A. Zaharia, A. Chandel, S. Saroiu, and S. Keshav, "Finding content in file-sharing networks when you can't even spell," in *IPTPS'07: International Workshop on Peer-to-Peer Systems*, 2007.

[3] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," 2001.

[4] O. Celma, "Music recommendation and discovery in the long tail," Ph.D. dissertation, Universitat Pompeu Fabra, Barcelona, 2008.

[5] A. S. Gish, Y. Shavitt, and T. Tankel, "Geographical statistics and characteristics of p2p query strings," in *IPTPS'07: International Workshop on Peer-to-Peer Systems*, 2007.

[6] "Ars Technica Report on P2P File Sharing Client Market Share, http://arstechnica.com/old/content/2008/04/study-bittorren-sees-big-growth-limewire-still-1-p2p-app.ars."

[7] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[8] R. J. Quinlan, *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, January 1993.

[9] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 1967, pp. 281–297.

[10] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *SCIENCE*, vol. 286, pp. 509 – 512, 15 Oct. 1999.

[11] 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 and the Netflix Prize Competition*, Las Vegas, USA, 2008.

[12] P. S. Bradley and U. M. Fayyad, "Refining initial points for k-means clustering," in *ICML '98*. San Francisco, CA, USA: Morgan Kaufmann, 1998.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[14] D. Knuth, *The Art of Computer Programming*. Addison-Wesley, 1973, vol. 3: Sorting and Searching.