

Mining Music from Large-Scale, Peer-to-Peer Networks

Yuval Shavitt, Ela Weinsberg, and Udi Weinsberg
Tel Aviv University

Using song files shared by users in the Gnutella network, the authors present a method for constructing a song-similarity graph to create scalable and efficient applications.

Millions of users worldwide use peer-to-peer (P2P) networks for sharing content, with a significantly high percentage of this content being multimedia, such as songs and movies.¹ As such, P2P networks are an invaluable resource for various multimedia information retrieval (MIR) tasks because of the large data set, the ability to capture data without the collaboration of P2P network operators, and a large and diverse population.

However, P2P networks are quite complex, exhibit high user churn, and contain high amounts of noise in the user-generated content. This makes collecting a complete snapshot of the network content complex. Additionally, there are often slightly different duplicates of the same files available in the network, which might have different file hashes, file names, and metadata tags. Duplication in metadata tags is typically caused by spelling mistakes, missing data, and different variations of the correct values. Finally, P2P user-to-item mappings are extremely sparse due to the vast amount of content, most of which is quite scarce, making user preferences hard to deduce.

These complexities result in difficulties when attempting to mine meaningful data from P2P networks. For example, even though improved search schemes² and recommender systems³ have been proposed to help users find content, current P2P networks mostly employ simple string-matching algorithms against

file name and metadata, either distributed or centralized, usually using a Web-based search engine. In the Gnutella network, this method results in only 7 to 10 percent of queries successfully returning useful content.⁴

While improving these approaches is obviously needed, recommender systems require meaningful data sets. Current recommender systems mostly rely on the willingness of users to rank their preferences to provide better recommendation. However, the nonexistence of explicit ranking in P2P networks and the previously mentioned complexities make it difficult to create efficient recommender systems, thus contributing to the increasing frustration of users. The main objective of the work described in this article is to overcome these difficulties and improve the ability to perform efficient mining of music content in data sets originating from P2P networks. For this project, we studied the musical content shared by users in Gnutella,⁵ then built a song-similarity graph, where the similarity between two songs is based on the number of users that share the two songs. We accounted for missing metadata by clustering the similarity graph and finding groups of similar songs. This article describes how the resulting clusters hold songs of varying popularity with high prevalence of dominant genres and artists, properties that are especially useful for recommender systems.

Song-similarity graph

Collecting the shared songs from a P2P network requires crawling the network, which involves traversing the network in a way similar to how Web crawlers behave. Using the Gnutella protocol, we can discover, for each crawled user, the set of peered users and the files they are sharing. We collected the data set used in this article in a 24 hour Gnutella crawl on 25 November, 2007. At this time, Gnutella was the most popular file-sharing network.⁶ The crawler reached more than 1.2 million Gnutella users and recorded more than 373 million files. Because this article focuses on MIR data, we identified files that are music-related by indexing only the files with a music-related suffix (MP3, WMA, FLAC, M4P, and M4A). We found that music-related content accounted for more than 75 percent of the files on the network, that is, more than 281 million files. These figures strengthen the notion that P2P

networks are an excellent source for MIR content, analysis, and applications.

We modeled the crawling data as a bipartite graph that connects users to files. This graph is a special case of the standard collaborative-filtering matrix in which a link in the graph represents the ranking of an item by a user. This graph is transformed into a song similarity graph, S , where the weight of a link between two songs is the number of users that hold both songs. Additionally, a popularity distribution vector, P , is created, counting the number of times each song appears in the network.

The similarity metric needs to be normalized to allow comparison of the similarities between pairs of files with different popularity. Each link weight w_{ij} is normalized using a modified cosine-distance function of the popularity of both songs given as follows:

$$\hat{w}_{ij} = \frac{w_{ij}}{\sqrt{P_i \cdot P_j}}$$

The normalized similarity graph is denoted by \hat{S} .

Intuitively, because some popular songs are shared by many users while many songs are shared by only a few, the graph should result in a power-law degree distribution. In such distributions, the probability that a randomly selected node is of degree k is $P(k) \approx k^{-\gamma}$, for a fixed $\gamma > 1$. This results in a few popular files with high degree and many less-popular files with low degree. In the following section, we show that, indeed, the degree distribution follows a power-law distribution.

Unique song identification

One of the difficulties caused by the noisy data set is the correct identification of unique songs. To this end, we use several techniques. First, songs having hashes that appear only once in the data set are filtered out, mainly because they are too scarce to provide any meaningful data. We then group all file hashes that relate to identical metadata value (artist and title), including artist and title values that have a small edit distance (counting insert, delete, and substitute), accounting for small spelling mistakes. Representative metadata values for each group are chosen using majority voting.

After this aggregation, all songs that have less than seven occurrences are removed. We chose this value as a trade-off between filtering and memory consumption. This song unification reduced the number of unique songs from more than 21 million to 530,000. Although this technique can result in over-filtering, it successfully overcomes the low signal-to-noise ratio that inherently exists in a P2P network, primarily due to user generated content. However, it's possible that some uses of P2P data would need more precise filtering techniques.

We perform additional filtering of weak song-to-song relations. Such links are mainly the result of heavy sharers, that is, users that share many different and uncommon mixtures of songs, contributing links between songs that are not commonly shared together. To this end, only links that appear in at least 16 different users are included, a value that we selected as a trade-off between filtering and resources. Then, we kept for each file only the top 40 percent of links (ordered by descending similarity value) and not less than 10. After this filtering, out of the original 50 million undirected links, roughly 20 million remain.

Partial sampling

Partial sampling is inherent to P2P networks, mostly because it's impossible to crawl all users and get all of their data, either due to the high churn or to security and privacy issues. However, partial sampling of the network is expected to lower all the links in the graph in the same proportion with the exception of the weakest links, where some links will disappear and some will have reduced weight. This partial crawl will not change the end results.

To evaluate the effect of partial sampling, we used a simpler, yet equivalent, approach. We created graphs, denoted by top relations (TR) N , where each graph contains, for each file, only the top N neighbors, ordered by non-increasing, normalized similarity. This technique captures the relative popularity of adjacent files, and is analogous to the effect of a partial sampling in the P2P network, where many users are simply not reached during the crawling phase. We will use these graphs to evaluate the way the similarity graph is affected by partial sampling. For additional analysis that we performed on these aspects, we refer the reader to Koenigstein et al.⁷

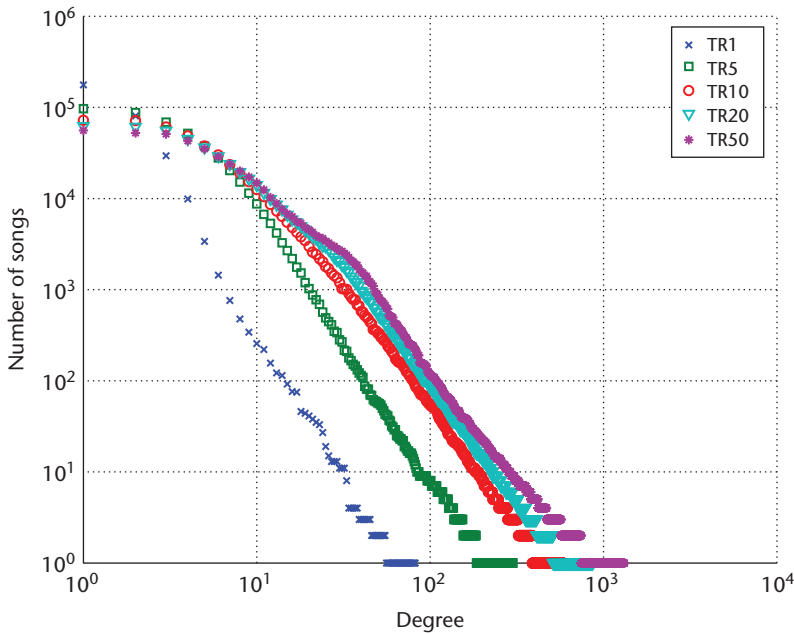


Figure 1. Degree distribution of the song similarity graph.

The degree distribution depicted in Figure 1 shows, as expected, a power-law distribution for all samples that have a similar exponent value. Furthermore, the figure shows that while for $N = 1$ the distribution is extremely sparse, for $N \geq 10$ we get an almost identical distribution with slight shifts. These results indicate that obtaining partial file-sharing information is sufficient for generating a comprehensive similarity graph, as the utility of having a more complete view of the network quickly diminishes.

Approximated search and recommendation

Users in P2P networks mostly search for content by using query strings that are matched against metadata fields attached to the content, either directly in the shared file or in a file's tracker. However, as stated before, metadata fields are often missing or quite ambiguous. For example, more than 35 percent of the files in our data set are missing a genre, while the remaining files have more than 3,600 different genres, some of which are encoded with numbers. This missing metadata obviously causes query-based searches to be rather ineffective in returning correct content.

Overcoming these difficulties is achieved by creating clusters of the shared songs on the basis of an estimation of the distance between shared songs. The resulting clusters can help interpolate missing metadata, therefore

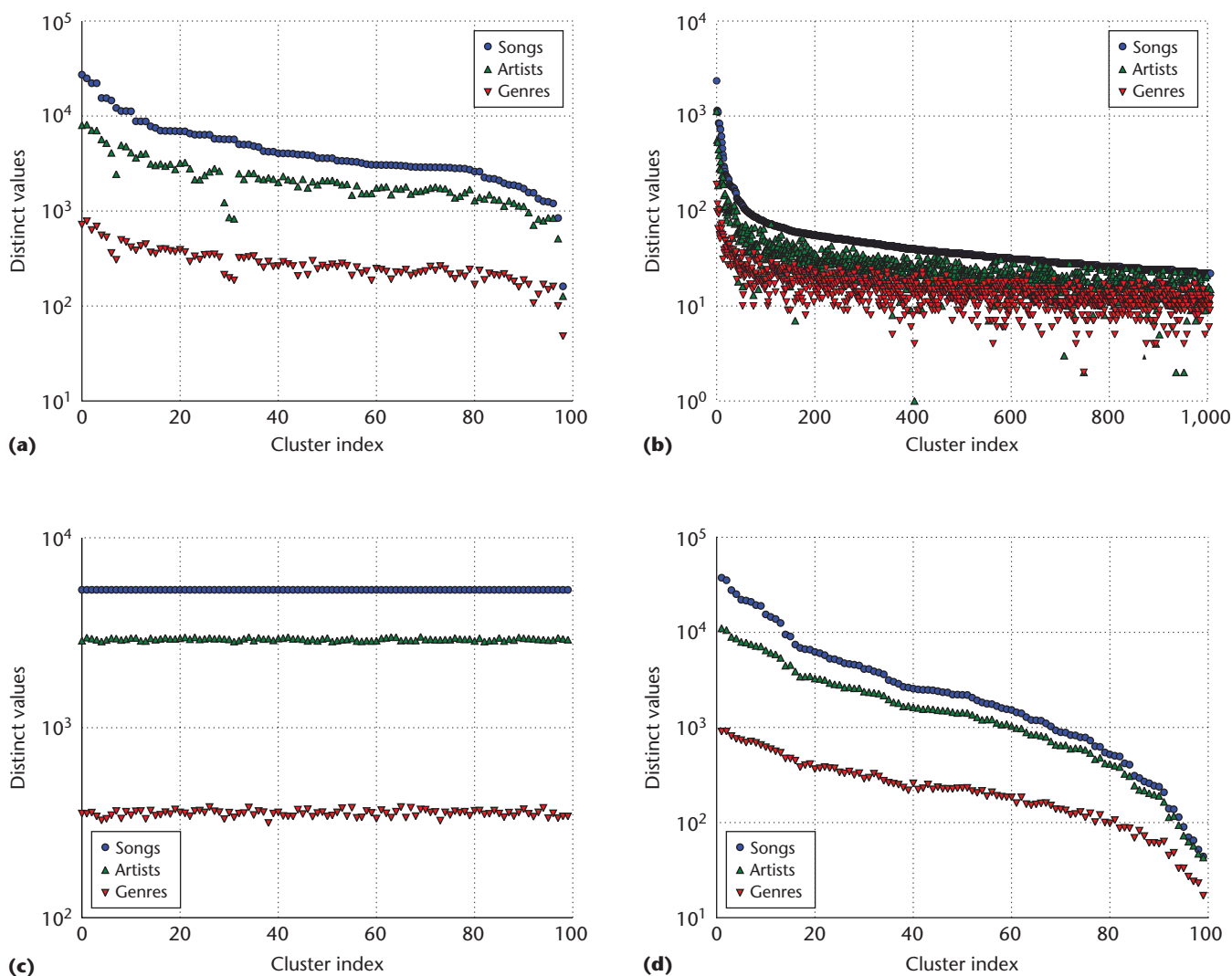
improving the ability to search for specific songs even if they have missing or misspelled metadata. Furthermore, recommender systems benefit by significantly scaling down the problem, estimating similarities using co-occurrences within clusters, and managing to detect less popular songs.

Clustering algorithm

To perform efficient clustering of the song-similarity graph, we use a variation of k -medoids,⁸ which is a distance-optimizing algorithm. The k -medoids method randomly selects k data points as cluster origins, then assigns each data point to the nearest medoid. It then shifts the medoids and reiterates the assignment process until the overall distance of each data point from the nearest medoid is minimal. This way, k -medoids is able to perform local optimization of these distances, hence is less affected by the distribution of the data points and quite robust to extreme data points and outliers. However, k -medoids doesn't operate directly on graphs, hence it requires its projection onto a multidimensional space. Moreover, k -medoids performs repeating iterations until converging, hence it doesn't scale. An in-depth review of clustering methods is available elsewhere.⁹

Our modified algorithm, called Graph k -medoids (GkM), includes several adaptations, making it more suitable to run on large-scale graphs. GkM operates directly on the similarity graph by calculating the distance between any two connected songs using a shortest-path algorithm, such as Dijkstra. Songs are assigned to the nearest cluster unless the distance to another cluster is roughly the same (up to some threshold). In the latter case, the song is assigned to the cluster with fewer songs, so that, overall, the number of songs in each cluster will be balanced. GkM performs a single-iteration medoid selection, allowing the algorithm to be highly scalable. Although this makes the algorithm not necessarily optimal, the vast size of the graph causes the algorithm to perform well.

GkM selects the cluster origins (medoids) once in a random method, keeping the distance between them larger than a preset value `min_dist`. Due to the random nature of this selection, and the lack of converging iterations, we might expect that the resulting clusters could vary significantly between runs.



However, the heuristic behind the algorithm design is that due to the large size of the graph, the `min_dist` limitation, and existence of popular and strongly connected vertices, clusters quickly grow toward popular vertices and extend from there. Pathological cases can rise when extremely remote and not well-connected vertices are selected as cluster origins. In these rare cases, however, these clusters will include only a few vertices and hence can be either discarded or manually identified to capture a unique set of similar vertices.

Clustering evaluation

To quantify the ability of the clusters to assist in search tasks and compare the results of several clustering algorithms, we present measures that capture how closely related the songs are in each cluster and how different the songs are between any two clusters. These questions play an

important role when searching for songs by identifying clusters in which to search or from which to recommend.

Evaluation is performed using the graph TR10 with $k = 100$. Converting similarity to distance is performed using $d(w) = -\log_2(w)$. GkM takes roughly 5 minutes to complete on a quad-core, quad-processor server with 16 Gbytes of memory.

Although a uniform cardinality (number of different files) distribution along the different clusters is desired, it's not likely to occur in music data because there are a few main streams and many niches. In these cases, the main stream clusters are expected to be much larger than the ones that represent a niche or some subgenre.

Figure 2a depicts the number of songs, distinct genres, and distinct artists in each cluster. The figure shows a relatively balanced

Figure 2. Number of songs and unique metadata fields in each cluster using GkM and state-of-the-art, large-scale clustering algorithms (clusters are ordered by descending number of songs):

(a) GkM, (b) Markov Cluster algorithm, (c) Metis, and (d) Graclus.

distribution among the clusters. The median cardinality is roughly 3,000 songs, whereas four clusters go beyond 20,000 songs. The lower the number of genres and artists relative to the number of songs, the more accurate the identification of musical preferences will be, hence the ability to approximate metadata improves. The average genre-to-song ratio is 0.07, and is significantly lower than the average artist-to-song ratio, which is 0.49. While the latter seems rather high, it's still better than the current state of the art as shown in Figures 2b, 2c, and 2d.

For the clusters to be efficiently used by recommender systems, it's important that each cluster contains files with large diversity of popularity. This allows a recommender system to recommend less popular content, which is often difficult to find. The resulting clusters hold a mixture of popular and unpopular songs. Furthermore, looking at the number of top popular songs in each cluster, we found that most clusters have at least one song from the top 2,500, and all of them have at least one from the top 5,000. As mentioned previously, this property is especially useful for recommender systems because common collaborative filtering techniques are based on like-minded users causing them to support popular tastes stronger than the unpopular. Finding less-popular content that might be of interest to users is therefore problematic as these are not easily tracked.

Having files in a cluster sharing various features is expected to be reflected in their metadata. For each cluster, we find the dominant genre and dominant artist (that is, the genre and artist that have the highest prevalence), and the percentage of their appearance out of all the files in the cluster that have valid metadata.

In roughly 60 percent of the clusters, more than 10 percent of the songs belong to the dominant genre, and in 5 percent of the clusters 30 percent of the songs belong to the dominant genre. These figures show that many of the songs in a cluster share a common feature (recall that there are more than 3,600 genres in our database). Additionally, in 94 percent of the clusters, the maximum prevalence of the dominant artist is less than 3 percent. However, noting that there are over 100,000 artists in the data set, this percentage is still quite high.

Next, we consider the number of additional significant genres and artists in each cluster. A significant genre or artist is defined as the one that has a prevalence of at least half the prevalence of the dominant genre or artist. On average, each cluster contains only two significant genres and four significant songs. Recalling that clusters contain thousands of songs, the metadata makes evident that songs residing in a cluster share common features.

Because many popular and productive artists create music that spans different genres, their metadata will have significant weight in several clusters. Additionally, although there are many different possible genres, most songs are tagged using only a handful of genres. Therefore, we might expect that several clusters will share the same dominant artist and the same dominant genre. This similarity, however, can be useful for recommender systems because it shows that clusters can reveal a segmentation of the files, which is significantly more fine-grained than that possibly obtained by user-generated tagging.

We analyzed the number of clusters in which each of the top-10 most dominant genres appear as dominant. We found that a few genres are dominant in many different clusters with up to 40 percent of the clusters having the same dominant genre. This finding shows that there can be many variations of the same genre that aren't properly tagged but are identified using cluster analysis. The distribution of dominant artists is, as expected, much broader than genres, where only a few artists appear as dominant in more than one cluster. However, it's still possible to detect different audiences for a given artist. For example, although Lil Wayne's songs are tagged with four genres (rap and hip-hop in our database and rock and pop-rap in Wikipedia), he appears in more than 10 different clusters, meaning that he has songs that spread across several different styles.

Comparison to state-of-the-art algorithms

We compare the results of GkM with several state-of-the-art unsupervised clustering algorithms. Markov Clustering (MCL) algorithm performs hierarchal clustering on the basis of stochastic flows and finds an optimal number of clusters.¹⁰ However, MCL has scalability issues, hence it's extremely slow and outputs many small-sized clusters (although some improvements were recently suggested).¹¹

Metis uses a recursive, multilevel, bisection-partitioning algorithm, resulting in a fast algorithm that tends to force a balanced partitioning of songs in each cluster.¹² Graclus attempts to optimize the normalized cut of the resulting clusters (that is, the ratio of outgoing links from each cluster to the overall cluster degree) using multilevel partitioning similar to Metis.¹³

A common problem with these algorithms is that they don't directly attempt to find vertices that are close to each other, but rather optimize efficiency metrics that are less suited to search and recommendation tasks, mainly due to the power-law nature of the similarity graph. As such, a balanced partition is desired but should not be strongly forced, making MCL, which creates thousands of small clusters, and Metis, which creates completely balanced partitions, not suited for our needs. Similarly, Graclus optimizes the normalized cut, which, as we show, results in extremely large clusters.

We evaluated these algorithms using the same TR20 graph and efficiency metrics. As expected, Metis and Graclus run extremely fast, and complete in roughly 20 seconds, whereas MCL, which attempts to find an optimal number of clusters on the basis of a given coarseness level, took almost three days to complete. This alone makes MCL problematic for clustering of large networks. Recall that GkM took roughly 5 minutes to complete on the same hardware.

In our results, which show the number of songs, the distinct artists, and genres in each cluster, MCL (see Figure 2b), as expected, outputs many small clusters (only the largest 1,000 clusters out of the 129,000 resulting clusters are presented). While it's possible to use only the T largest clusters, the tail is so long that unless T is extremely large, most files will be lost. Metis exhibits an almost perfectly balanced 5,318 songs per cluster (see Figure 2c). Graclus has a large variety of cluster sizes (see Figure 2d). However, almost 30 percent of the clusters contain less than 1,000 songs, making them rather useless for our needs.

The metadata distribution in Metis is extremely smooth, which is unexpected because Metis balances the number of songs in each cluster without any knowledge about metadata. However, using a completely uniform distribution with the balls-in-bins formula provides similar results given that songs are grouped

into batches of between 10 and 12. This result can be explained by the strong relationships between songs that are in the same album (containing 10 songs on average), and hence are commonly downloaded together. Although Metis performs quite well, GkM provides better results with an average artists-to-songs ratio of 0.49 as opposed to the 0.55 of Metis. Furthermore, because GkM relaxes the size balancing constraint, it generates more natural clustering, which also includes much larger clusters than Metis. Graclus and MCL perform worst, because they both have too many small clusters, but still each have a variety of genres and artists, making them useless for our tasks.

The popularity of songs in Metis clusters is also well-balanced, while again Graclus and MCL perform badly. This result is due to Metis performing an almost uniform distribution of strongly connected songs into clusters. Because Graclus and MCL produce many small clusters, they are unable to provide high diversity of popularities, causing only the few large clusters to be effective.

Overall, GkM outperforms these state-of-the-art clustering algorithms using the studied metrics, indicating that it will most likely perform more effectively in search and recommendation tasks on power-law networks. Obviously, when comparing the optimized feature, for example, normalized cut in GkM and Graclus clusters, GkM falls far behind. Recalling that the clustering is not the goal, but rather the means toward recommending songs, this comparison shows that commonly used efficiency metrics are not always the best measure when considering specific applications of the resulting clusters.

Recommendation system

As a real world application, the song clusters are used for creating a recommendation system that provides users with songs that are related to the ones they already share. Because people usually have a defined taste in music, we expect that the songs of each user will reside in a small number of clusters with a large fraction of the songs in one of the clusters. This expected behavior provides the ability to identify a user's musical taste by mapping each user to her dominant cluster or even to several clusters. Recommender systems can take advantage of the diversity of file popularity within each cluster and recommend new and yet unpopular

content (of high quality), a task that is known to be difficult.¹⁴

Counting the number of songs a user has in each cluster revealed that 11 percent of the users have all their songs in a single cluster and almost 70 percent are mapped to fewer than 10 different clusters. Furthermore, the median of song prevalence in the user's dominant cluster (that is, the cluster that has the most songs) is almost 70 percent.

We create user recommendations with three methods. The first, which is used for comparison, simply recommends the overall most popular songs that a user doesn't have, in descending order of popularity. The second method improves on the first by finding the user's dominant cluster and recommending the most popular songs from that cluster. The third method considers all the songs neighboring the known songs in the dominant cluster, and iteratively recommends the nearest neighboring song. We refer to these methods as "most popular," "cluster popular," and "nearest neighbor" respectively.

We evaluate the results by randomly selecting 30 percent of the songs of each user. Then, each recommender system attempts to recommend the remaining 70 percent of the songs. We resolve the artists of the recommended songs and count the precision (that is, how many were correctly recommended out of the recommended set) and recall (that is, how many were correctly recommended out of the real data). Clustering was performed on the TR20 using GkM with $k = 100$.

Using the most popular method resulted in an average precision (taken over all users) of 19.4 percent and a recall of 18 percent. Cluster popular and nearest neighbor gave slightly better results with an average precision of 21.7 percent and 21.5 percent, and average recall of 18.7 percent and 18.1 percent, respectively. While these results seem somewhat low, they are quite good considering the vast amounts of songs that exist on P2P networks. For example, one user in our data set had songs tagged with "Bob Dylan featuring Van Morrison," "Chuck Berry," and "Bob Dylan." Given that only one song was tagged with "Bob Dylan ft. Van Morrison," the nearest neighbor algorithm recommended songs tagged with "Van Morrison." Although this is clearly a good recommendation, it was tagged as a failure by our automatic evaluation.

The popularity-based methods are successful due to the many users that prefer popular mainstream music. Furthermore, a major advantage of these methods is that preparing the list of possible recommendations can be performed for all users, while the latter requires processing per user. However, the latter can be refined so that it provides diversity in the recommendation, such as recommending the song that has smallest average distance to all other songs, the song that is nearest neighbor in one of the nondominant clusters or for more adventurous users, recommending the nearest neighboring song that is the least popular. Such recommendations, however, are much more subjective and thus are difficult to assess.

This analysis shows that it's possible, yet not trivial, to recommend the songs and artists that a user might prefer. Moreover, devising an automatic criteria for accurate estimation of the correctness of recommendations is a complicated task.

User similarity

A different use of the song-similarity graph is estimating the similarity between peers, making it possible to find like-minded users. For traditional search-string propagation, users that are similar to searching users are more likely to have the searched content than other users. In recommendation systems, it's obviously more promising to recommend content from like-minded users. However, developing user similarity metrics in modern P2P networks is challenging, mainly due to sparseness, meaning the overlap between users' shared songs is extremely small. In our data set, almost 85 percent of the users share less than 20 songs, while fewer than 3 percent of users share more than 50 songs.

Peer distance estimation

Calculating the distance between users using the songs-similarity graph manages to capture the wisdom of the crowd, as it estimates the distance between files on the basis of the global preferences of many users. Using the song-similarity graph, it's possible to calculate the distance between users using all of their shared files, and not only the mutually shared. For any two given users, we create a bipartite graph B that contains the songs of each user in each side. Each song is connected to songs of the

other user with a link weight, which is the shortest-path distance between the two songs on the similarity graph.

Once the bipartite graph is built, a maximum weighted bipartite matching algorithm is applied. To compare between users i and j that have different number of files, $|p_i|$ and $|p_j|$, the weight of the matching (sum of the link weights in the matching) is normalized using $\min\{|p_i|, |p_j|\}$, which is the average link weight. This value is used as the peer similarity.

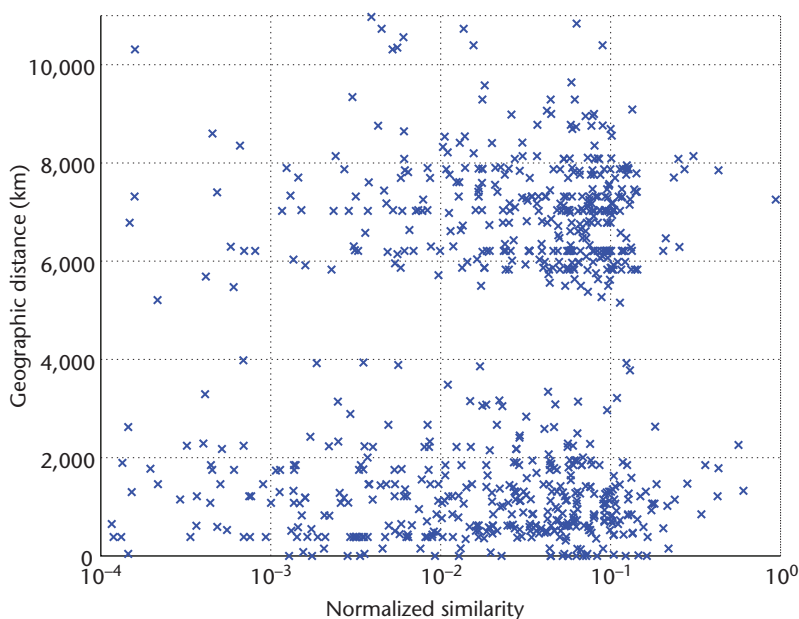
There are two main benefits of using maximum weighted matching. First, it provides an efficient method for assessing the best match between two sets of files, hence the approach provides a good estimation for the user similarity without requiring overlap of the shared files. Second, because only the highest-similarity links are selected to the matching, various filters can be used for reducing the size of the similarity graph and improving run time and memory consumption of the algorithm, with minimal bias in the results.

Results

Validation of the similarity metric is performed using the sampled set of 100,000 users and the TR10 song-similarity graph. For each pair of users, we compare the resulting similarity to the artist similarity. To accomplish this comparison, the artists performing the songs shared by each user are taken from the metadata fields. Assuming that two users i and j have two sets of artists A_i and A_j , the artist similarity is defined as $(|A_i \cap A_j|)/\min\{|A_i|, |A_j|\}$.

Figure 3 shows that high user similarity indicates high artist similarity, which validates the overall correctness of the similarity metric. However, high user similarity also exists when artist similarity is zero, showing that comparing exact songs between users, even using coarser granularity, is insufficient.

We have found that there is only a weak correlation between geographical distance and user similarity, indicating that spatial locality of interest¹⁵ is becoming less valid in P2P networks. As such, bootstrapping based solely on shared-content correlation or geographic locality is insufficient. Adding our proposed user similarity metric manages to more effectively include the wisdom of the crowd into the process of user-similarity estimation.



Conclusion

Overall, P2P networks provide an abundance of information that can be used in MIR research and can help produce exciting applications. However, data sets extracted from P2P networks should be carefully processed to fully harness its strengths. Careful analysis and filtering of the data set is key for sufficient noise reduction while maintaining relevant, useful, and representative information.

MM

Acknowledgments

We thank Tomer Tankel for providing the data used in this article, and sharing valuable ideas for its successful analysis. This research was supported in part by the Israel Science Foundation center of excellence program (#1685/07) and by the Ministry of Trade and Industry, Magnet program through the NeGeV Consortium.

References

1. A.S. Gish, Y. Shavitt, and T. Tankel, "Geographical Statistics and Characteristics of P2P Query Strings," *Proc. Int'l workshop on Peer-to-Peer Systems (IPTPS)*, USENIX Assoc., 2007.
2. B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS)*, IEEE CS Press, 2002.
3. P. Resnick and H.R. Varian, "Recommender Systems," *Comm. the ACM*, vol. 40, no. 3, 1997.
4. M.A. Zaharia et al., "Finding Content in File-Sharing Networks When You Can't Even Spell,"

Figure 3. Comparing user similarity with artist similarity (500 random users are shown for brevity).

- Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS), USENIX Assoc., 2007.*
5. M. Ripeanu, *Peer-to-Peer Architecture Case Study: Gnutella Network*, tech. report TR-2001-26, Univ. Chicago, 2001.
 6. E. Bangeman, *Study: Bittorrent Sees Big Growth, Limewire Still #1 P2P App*, Ars Technica report; <http://arstechnica.com/old/content/2008/04/study-bittorren-sees-big-growth-limewire-still-1-p2p-app.ars>.
 7. N. Koenigstein et al., "On the Applicability of Peer-To-Peer Data in Music Information Retrieval Research," *Proc. Int'l Society for Music Information Retrieval Conference (ISMIR)*, 2010.
 8. R.T. Ng and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB)*, Morgan Kaufmann Pub., 1994.
 9. A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 31, no. 3, 1999, pp. 264-323.
 10. S.V. Dongen, *Performance Criteria for Graph Clustering and Markov Cluster Experiments*, tech. report, Nat'l Research Inst. for Mathematics and Computer Science, 2000.
 11. V. Satuluri and S. Parthasarathy, "Scalable Graph Clustering Using Stochastic Flows: Applications to Community Discovery," *Proc. 15th Conf. Knowledge Discovery and Data Mining (KDD)*, ACM Press, 2009.
 12. G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *Proc. Int'l Conf. Parallel Processing*, IEEE CS Press, 1995.
 13. I.S. Dhillon, Y. Guan, and B. Kulis, "Weighted Graph Cuts without Eigenvectors a Multilevel Approach," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, 2007, pp. 1944-1957.
 14. J.L. Herlocker et al., "Evaluating Collaborative Filtering Recommender Systems," *ACM Trans. Information Systems*, vol. 22, no. 1, 2004, pp. 5-53.
 15. S. Voulgaris et al., "Exploiting Semantic Proximity in Peer-To-Peer Content Searching," *Proc. 10th Int'l Workshop Future Trends in Distributed Computing Systems*, IEEE CS Press, 2004.

Reach Higher

Advancing in the IEEE Computer Society can elevate your standing in the profession.

- Application in Senior-grade membership recognizes ten years or more of professional expertise.
- Nomination to Fellow-grade membership recognizes exemplary accomplishments in computer engineering.

GIVE YOUR CAREER A BOOST


UPGRADE YOUR MEMBERSHIP

www.computer.org/join/grades.htm

Yuval Shavitt is a faculty member in the School of Electrical Engineering at Tel-Aviv University, Israel. His research interests include Internet measurements, mapping, and characterization; and data mining peer-to-peer networks. Shavitt has a D.Sc. in electrical engineering from the Technion, Haifa, Israel. Contact him at shavitt@eng.tau.ac.il.

Ela Weinsberg is an MS student in the department of industrial engineering at Tel-Aviv University, Israel. Her research interests include data mining of peer-to-peer networks. Weinsberg has a BS in computer science and mathematics from Bar-Ilan University. Contact her at ela@eng.tau.ac.il.

Udi Weinsberg is a PhD candidate in the school of electrical engineering at Tel-Aviv University, Israel. His research interests include Internet measurement, complex networks analysis, and large-scale data mining. Weinsberg has an MS in electrical engineering from Tel-Aviv University, Israel. Contact him at udiw@eng.tau.ac.il.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.