Knowl Inf Syst
DOI 10.1007/s10115-013-0619-9

REGULAR PAPER

Peer-to-peer information retrieval using shared-content clustering

Irad Ben-Gal · Yuval Shavitt · Ela Weinsberg · Udi Weinsberg

Received: 13 July 2011 / Revised: 6 February 2013 / Accepted: 17 February 2013
© Springer-Verlag London 2013

Abstract Peer-to-peer (p2p) networks are used by millions for searching and downloading content. Recently, clustering algorithms were shown to be useful for helping users find content in large networks. Yet, many of these algorithms overlook the fact that p2p networks follow graph models with a power-law node degree distribution. This paper studies the obtained clusters when applying clustering algorithms on power-law graphs and their applicability for finding content. Driven by the observed deficiencies, a simple yet efficient clustering algorithm is proposed, which targets a relaxed optimization of a minimal distance distribution of each cluster with a size balancing scheme. A comparative analysis using a song-similarity graph collected from 1.2 million Gnutella users reveals that commonly used efficiency measures often overlook *search and recommendation* applicability issues and provide the wrong impression that the resulting clusters are well suited for these tasks. We show that the proposed algorithm performs well on various measures that are well suited for the domain.

Keywords Peer-to-peer · Data mining · Recommender systems · Clustering

1 Introduction

Peer-to-peer (p2p) networks are used for sharing content (i.e., files) by millions of users worldwide (note that we use the term content to refer to the files that are shared by users and not to the bits that comprise the files themselves). Users mostly search for content by using query strings which are matched against metadata fields that are attached to the content,

I. Ben-Gal · E. Weinsberg
Deparment of Industrial Engineering, Tel-Aviv University, Tel-Aviv, Israel

Y. Shavitt
School of Electrical Engineering, Tel-Aviv University, Tel-Aviv, Israel

U. Weinsberg (✉)
Technicolor, Palo Alto, CA, USA
e-mail: udiw@eng.tau.ac.il

Published online: 09 March 2013

either directly in the shared file or in a file's tracker. For example, song files contain metadata (ID3 tags) that describe the name of a song, genre, artist, and album.

Often, some of this data are missing, incorrectly spelled, or encoded (such as musical genre), making it difficult for users to find the data they are looking for in the abundance of existing content. As such, even though improved search schemes [27,51] and recommender systems [34] were proposed to help users find content, current p2p networks mostly employ simple string matching algorithms against the file name and its metadata. These algorithms are either distributed or centralized, usually using a web-based search engine. For example, in the Gnutella network, this scheme causes only 7–10 % of the queries to successfully return useful content [52].

Several difficulties arise when trying to improve the efficiency of search schemes and recommender systems that operate on such networks. These include implicit ranking, i.e., lack of explicit ranking of content by users, the large amount of noise which is attributed to user-generated content, the power-law structure of the network, and the extreme sparseness of the network which is the result of the abundance of users and shared content. These complexities often result in frustration of p2p users that are unable to find content using standard search methods, whereas recommender systems that aim to help users find new unfamiliar content cannot efficiently operate in these networks.

This paper studies the search and recommendation applicability of clustering power-law graphs in general and its implications to p2p networks in particular. We first study the problems that rise when applying existing clustering algorithms on power-law graphs. We validate the applicability of clustering by using a real-world recommender system. The usefulness in generating recommendations with significantly smaller search dimensions is evident in this case. Then, a simple yet efficient clustering algorithm, named Graph k-medoids (GKM), is presented, targeting a relaxed minimal distance optimization.

The algorithm is simple, yet scalable. It uses the distance between co-shared files as an item-similarity metric, mitigating the need for in-depth bit-level analysis of the files. Since it is based on the well-studied k-means algorithm, it benefits from all the intensive research already performed on k-means, such as distribution [21], measure analysis [50], and user-centered privacy [36].

In order to provide insightful conclusions on search and recommendation domain, evaluation is performed on a real-world graph, containing over 530k songs shared by roughly 1.2 million users in the Gnutella [35] p2p network. The usefulness of p2p data for data-mining applications is presented alongside with a study of its inherent difficulties.

Using common and domain-specific efficiency measures, an extensive comparative analysis is performed. It shows that GKM obtain better results than other state-of-the-art clustering algorithms when applied to a power-law graph.

The empirical analysis shows the robustness of the proposed clustering algorithm to the selection of various input parameters, and a comparative analysis emphasizes its strengths and weaknesses over existing state-of-the-art methods. We show that commonly used efficiency measures often overlook search and recommendation applicability issues and provide the wrong impression that the resulting clusters are well suited for these tasks, whereas the proposed algorithm manages to perform well on all measures.

The contribution of this work is threefold. First, the difficulties of applying state-of-the-art clustering algorithms on power-law graphs are presented. Then, a simple relaxed distance optimization algorithm is proposed that aims at improving existing algorithms when applied to power-law graphs. Finally, using a real-world large-scale power-law graph, an extensive comparative analysis of clustering algorithms is performed, showing that commonly used efficiency measures are insufficient for reflecting the applicability of clusters for search and recommendation tasks. Using domain-specific efficiency measures, we show that overall distance-optimizing algorithms perform better than current state-of-the-art ones.

2 Related work

The popularity of p2p multimedia file sharing applications, such as Gnutella, has created a flurry of research works related to p2p architectures. Concerns were raised regarding the ability to extract useful information from such networks. Saroiu et al. [37] described the need for accurate peer characterization in order to correctly evaluate a p2p system; specifically, they address questions such as how many files are shared and downloaded by the peers. Luo et al. [25] studied the problem of performing distributed classification in p2p networks and showed that although distributed classification was extensively studied, p2p networks give rise to new challenges, such as peer churn and frequent data updates.

Several studies on p2p networks [13,16] show that graphs that model various p2p networks exhibit power-law distributions. Stutzbach et al. [46] presented a detailed characterization of topology that is typical in modern popular p2p systems and examined different forms of graph analysis and their properties. The authors raised a concern that power-law degree distributions could be a result of measurement artifacts, and presented [45] methods for sampling various peer properties, e.g., degree, link bandwidth, number of files shared, that result in nearly unbiased samples under a wide variety of commonly encountered p2p network conditions.

In this work, we analyze various samples of graphs extracted from p2p networks. We show that given sufficiently extensive and representative sampling process, the obtained graph, which is constructed from files shared by peers, is robust and exhibits power-law behavior, even in the presence of partial sampling.

Mining structural data from large-scale graphs is an active research topic [22]. Clustering a graph is a process that partitions the vertices of the graph into clusters [20], such that vertices in the same cluster are “similar” in some sense, while vertices in different clusters are “not-similar,” respectively. Several existing graph clustering algorithms optimize some aspect of the resulting clusters (for a review, see [44]).

In this paper, we focus on several popular graph clustering algorithms. In particular, we compare the results of the GKM to those obtained from several state-of-the-art clustering algorithms. Some of the difficulties of these clustering methods are known. For example, MCL [11] performs hierarchal clustering based on stochastic flows; however, it is known to have scalability issues [38]; hence, it can be extremely slow and can output many small-sized clusters. Graclus [9] minimizes the commonly used clustering metric of normalized cut of the resulting clusters, i.e., the ratio of outgoing links from each cluster to the overall cluster degree, using multilevel partitioning. Metis [23] is a similar hierarchal clustering algorithm, which adds a constrain of producing a balanced partitioning of vertices in each cluster. We elaborate on these algorithms and their outcome in Sect. 3.2.

Searching in p2p networks was extensively studied. Various techniques were proposed, such as approximate searching [26,27,29,51], interest locality [43], semantic overlays [47], and even complete restructuring of the network [49]. However, most p2p networks still employ the simple string matching technique against the name of the file or its metadata, if exists. However, this simple technique was shown to perform poorly [52].

Clustering the shared content in p2p network was proposed [13,24,41,42] to have the potential for overcoming some of the above-mentioned difficulties. However, the structure of p2p networks as well as many other networks [3] often follows a power-law degree distribution. This causes existing clustering algorithms to fail to produce clusters that are well suited for efficient search and recommendation content.

Recommender systems have been extensively studied in the past in various contexts and applications. Zheng et al. [53] described an approach to identify peers that share similar music preferences and cluster them into communities. Anglade et. al [2] presented a technique for clustering peers according to their explicitly or implicitly stated music preferences. They showed high percentage of of successful assignment of peers into their communities. However, it has been claimed that accurately characterizing peer preferences [45] and finding a distance metric between peers are far from being trivial [40]. For a thorough review on recommender system or on music recommendation, the reader is reffered to [34].

Note that although we present a simple recommender system built on top of clusters, the focus of the paper is not on building a novel recommender system. Rather we seek to show the applicability of shared file clusters to recommendation. We address the ability to match search strings to actual content when metadata are missing, and the ability to scale down the vast dimensions of searching and recommending problems in p2p networks.

3 Methodology

Clustering the content of a p2p network starts with collecting the list of files shared by each user. Once collected, the data are processed into a similarity graph that is used as the input for the clustering algorithm which, upon completion, maps the collected files into clusters.

3.1 Definitions

The initial input for the problem is a matrix, denoted by F, that serves as an indicator to the files that users share:

$$F(i, j) = \begin{cases} 1 & file\ j \in user\ i \\ 0 & otherwise \end{cases}$$

The matrix is obtained by crawling the network and collecting all the files that a user shares. Since the number of users and different files is often extremely large, the network is very sparse. Notice that this matrix is a special case of the more standard collaborative filtering matrix, yet instead of rankings we have only an indicator.

Based on this matrix, a file similarity matrix S with size $m \times m$ is created using:

$$S = F^T F$$

where each entry $S(i, j)$ counts the number of users that share both file i and file j.

Additionally, a popularity distribution vector $P \in \mathbb{R}^m$ (number of files) is created, counting the number of different users that share each file:

$$P(i) = \sum_{j=1}^{n} F(i, j)$$

These vectors are used later to normalize the graph weights. In practice, performing such a scalar multiplication using a matrix with 1.2 million rows and over half a million columns (as we present in Sect. 4) is almost impossible with commodity hardware. Therefore, in order to obtain this graph, we go over each user and gradually build pairs of files that are shared together using 4 bit counters. This procedure enables relatively low usage of memory for pairs that do not appear often together. Only for a pair that exhausts the counter, we accommodate a new 4 byte counter and reuse the previous 4 bit counter for other pairs. This graduate and conservative memory consumption enables building the similarity matrix and popularity vector using commodity hardware in reasonable time.

For simplicity reasons, we refer to the similarity matrix S as a similarity graph, where each node is a file, and the weight of a link between two files represents the similarity metric between them. Given two nodes i and j, the weight of the link w_{ij} connecting these nodes is simply the number of users that share both file i and file j.

This similarity metric is normalized to allow comparison of similarities among pairs of files with different popularity. Each link weight w_{ij} is normalized using a modified cosine-distance function of the popularity of both files:

$$\widehat{w}_{ij} = \frac{w_{ij}}{\sqrt{P(i) \cdot P(j)}} \tag{1}$$

Since w_{ij} counts the number of different users sharing files i and j together, then $w_{ij} \leq P(i) + P(j)$ (equality is obtained when the two files are always shared together). Obviously, $w_{ij} \leq P(i) \cdot P(j)$, and therefore, the normalized similarity metric, denoted by $\widehat{S}$, holds $0 < \widehat{w}_{ij} \leq 1$.

3.2 Existing graph clustering algorithms

Intuitively, since some (relatively few) popular files are shared by many users while most of the files are shared by only a few users, the graph results in a power-law [3] degree distribution. In such distributions, the probability that a randomly selected node will be of degree k is $P(k) \simeq k^{-\gamma}$, for a fixed $\gamma > 1$. This results in high degree of the few popular files and lower degree of many less-popular files. We validate this intuition in Sect. 4 and show its implications on the clustering algorithms.

The power-law distribution has a major downside as it prevents common state-of-the-art clustering algorithms from achieving a graph partitioning that is suitable for searching content in the p2p network. In particular, in many cases, the power-law distribution causes state-of-the-art clustering to produce biased clusters. On the other hand, as we later show, it enables applying efficient distance-optimizing clustering algorithms. Moreover, in this network, even partial view of the network is sufficient to efficiently capture many of the underlying graph properties.

MCL [11] performs hierarchical clustering based on stochastic flows and finds the optimal number of clusters. The intuition behind MCL is that using random walks based on vertex transition probabilities (i.e., link weights) allows the algorithm to identify all the vertices that flow to the same "attractor" vertex, hence belong in the same cluster. While MCL is useful in domains that use relatively small graphs, it has been shown [38] to be slow due to scalability issues and to output many small-sized clusters. While the first feature is clearly problematic for applying on large graphs, the latter is also undesired for search and recommendation tasks since the clusters break related content, inducing a spatial search over the set of clusters which is not much smaller than the original problem size.

A commonly used efficiency measure for evaluating clustering algorithms is the normalized cut (or *conductance*). The normalized cut is the percentage of links that connect vertices within a cluster to vertices outside the cluster. Denote by $A(i, j)$ the adjacency matrix representing connectivity of graph G. The normalized cut of a cluster C is defined using:

$$Ncut(C) = \frac{\sum_{i \in C, j \notin C} A(i, j)}{\sum_{v_i \in C} degree\,(v_i)} \tag{2}$$

Graclus [9] is a clustering algorithm that optimizes the normalized cut of the resulting clusters while using multilevel partitioning. Although the normalized cut measure was proven to be quite efficient, applying it on a power-law graph is problematic. In power-law graphs, high-degree vertices are strongly connected to both many low-degree vertices and other high-degree vertices, i.e., popular content is connected to both popular and less-popular contents. Assuming that each of two distinct clusters contain only a single highly popular file, both will have very high cut, driven by the many less-popular files that are connected to each. Given that one of these clusters will "pull" most less-popular files, the second cluster, containing the other very popular file will have a very high cut, since it is also connected to many of the same less-popular files, and with high probability to the other popular file. Hence, the best cut will be achieved by merging the two clusters into one. This process will lead most files to be pulled into a single cluster, resulting in a "mega" cluster. This is undesired as the clusters do not significantly reduce the dimensions of the original problem in our considered case. We note that some improvements to Graclus have been proposed, e.g., combining it with other clustering algorithms [28]; however, these mostly target avoiding breaking up well-connected communities, which is not an issue with our data set.

Metis [23] is one of the most well-known multilevel partitioning algorithm that uses coarsening and refinement heuristics for achieving a fast running algorithm. In addition to an objective of minimizing the normalized cut, Metis also enforces a constraint of producing equal-size partitions (any two clusters have roughly the same number of vertices). Although this constrain avoids the problems in Graclus, by enforcing smooth partitioning into equal-size clusters, Metis breaks large groups in an unnatural fashion. Since it is based on the normalized cut, it mainly enforces small strongly connected subgroups to remain in the same cluster, while shifting weaker connected groups arbitrarily to other clusters. Similarly, clusters that should be kept small are unnaturally merged with other small groups, so that the cluster size remains the same.

The k-medoids algorithm [30] is another distance-optimizing algorithm, which is similar to the well-known k-means algorithm except that it uses data points as cluster "centroids". Since it operates on a multi-dimensional feature space and not on a graph, it requires the projection of the graph onto this space [39], hence creating a dense matrix of distances between vertices. Platt [32] showed that it is possible to obtain clusters of musical genres using fast sparse embedding, a variation of the FastMap [12] embedding algorithm. However, projecting the already known distances between files onto a multi-dimensional space results in a distance approximation, which serves as an approximation of the original distance, and thus results in distortion of the data. When the projected graph is extremely large with very high degrees, as expected to be the case of the p2p file similarity graph, the distortion can be quite severe, causing an aggregated mistake and leading to a biased clustering. A different approach that approximates k-mean was presented in [48]; however, it was mostly experimented with small-scale data sets (thousands of items).

3.3 Relaxed distance optimization

Accounting for the above, we design a new clustering algorithm, which is based on the concepts of k-medoids. Similar to k-medoids, our proposed algorithm, denoted by GKM, randomly selects k data points as cluster origins and then assigns each data point to the nearest medoid.

There are several major differences between k-medoids and GKM. First, k-medoid shifts its selected set of medoids toward a local minima and then reiterates the assignment process until the overall distance of each data point from the nearest medoid is minimal. This way, *k*-medoid 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. On the other hand, repeating these iterations until convergence prevents the algorithm from scaly well. Due to the extreme graphs that GKM is designed to operate on, GKM does not perform iterations. We later discuss the implications of this design aspect.

Second, *k*-medoids do not operate directly on graphs; hence, as stated above, this requires the projection of the data points onto a multidimensional space. While there are some methods for projection that introduce little bias when projected on large dimensions (see [14] for a survey of such methods), operating directly on the graph completely avoids this bias altogether, by preserving the original distances between vertices in the graph.

Finally, the selection of the *k*-medoids is commonly done in a completely random fashion, counting on the iterations to fix incorrect selections. This random selection process often leads to incorrect clustering of the data points; hence, some methods for improving the initial selection were suggested in the past [7,19,31]. We take a different approach and tackle this problem by selecting medoids that are sufficiently distant from one another. This aspect is detailed in the next sections.

Conceptually, GKM operates in a similar fashion to *k*-medoid as it attempts to minimize the total distance of vertices in each cluster from the "center" (or *origin*) of the cluster. More formally, given a set O of k preselected cluster origins, the primary algorithm optimization goal is as follows:

$$\min \sum_{i=1}^{k} \sum_{v \in C_i} d\left(v, O_i\right) \tag{3}$$

We note that selecting a "good" value for k is a nontrivial task, mainly since it is highly domain specific, i.e., it should somehow capture the meaning behind the clusters that are generated from the data set. The evaluation of data in this paper represents musical preferences of users, and the clusters are supposed to capture songs that have similar features. In such subjective domains, precisely capturing the number of clusters is unlikely; however, as we later show, the evaluated applications are not sensitive to the exact selection of k, given that it is large enough.

The algorithm receives as input a similarity graph, $G(V, E)$, the number of output clusters, k, and a size balancing threshold τ, which is used to balance the size of the resulting clusters as explained later. Since the algorithm operates on distances rather than similarity, we assume the existence of a function $d(v_1, v_2)$ that calculates the distance between any two (connected) vertices. This is achieved by converting the similarity metric into a distance metric and summing distances along the shortest path between the vertices (e.g., using Dijkstra [10]).

The algorithm pseudocode is shown in Algorithm 1. The algorithm operates in the two steps listed in lines 4 and 5—it first finds a set of cluster origins using the FINDCLUSTERORIGINS procedure and then uses a single iteration to assign the vertices to the nearest cluster using the ASSIGNVERTICESTOCLUSTERS procedure. The remaining code handles the case of a graph which is not fully connected, and contains a set of connected components (*cc*). In this case, each component *cc* is separately clustered to k_{cc} clusters, where k_{cc} is computed using the proportion of the numbers of vertices contained by the connected component.

If the size of the connected component is too small, the algorithm simply adds it as a complete cluster. Note that in a highly fragmented graph, this results in more clusters than the provided k. To avoid this, it is possible to simply throw small clusters, i.e., remove line

Algorithm 1 GKM

```
Require: Graph $\widehat{G}(V, E)$, number of clusters $k$, distance function $d(v_i, v_j)$, size balancing threshold $\tau$
Ensure: Cluster set $C$
1: for all $cc(V_{cc}, E_{cc}) \in Connected\_Components(G)$ do
2:     $k_{cc} \leftarrow \lfloor k \cdot |V_{cc}| / |V| \rfloor$
3:     if $k_{cc} > 1$ then
4:         $O \leftarrow$ FindClusterOrigins$(V_{cc}, k_{cc}, d)$
5:         $C \leftarrow$AssignVerticesToClusters$(V_{cc}, O, d, \tau)$
6:     else
7:         $C \leftarrow C \cup V_{cc}$
8:     end if
9: end for
10: Return $C$
```

7 in the code above; however, we assume that the exact value of k is less important in terms of performance than actually extracting the correct clusters.

3.3.1 Finding cluster origins

Algorithm 2 shows how the set of k cluster origins is selected. Basically, the origins are selected at random, such that there is a minimal distance between every two origins, i.e., $\forall i, j \in \{1 \ldots k\}, i \neq j, d(O_i, O_j) > min\_dist$. The min_dist value is obtained using the ApproxMinimalDistance(V, k, d) procedure which is explained later.

GKM selects a random vertex, which is the origin of the first cluster, O_1. Then, the next origin O_2 is selected, making sure that the distance $d(O_1, O_2)$ is at least min_dist. If the distance is found to be smaller, a new vertex is randomly selected and the test is performed again; otherwise, GKM proceeds to the next origin. This selection process is repeated until k origins that satisfy the minimal distance constraint are found.

Algorithm 2 FindClusterOrigins

```
Require: Vertex set $V$, number of clusters $k$, distance function $d(v_i, v_j)$
Ensure: Cluster origin set $O$
1: $min\_dist \leftarrow$ApproxMinimalDistance$(V, k, d)$
2: $i \leftarrow 0$
3: while $|O| < k$ do
4:     $O_i \leftarrow Select\ v \in V\ uniformly\ at\ random$
5:     if $\forall o \in O \setminus O_i, d(O_i, o) \geq min\_dist$ then
6:         $i \leftarrow i + 1$
7:     end if
8: end while
```

Since a fast running algorithm is usually more important than memory consumption, it is possible to store the distances from each origin O_i to all other vertices in the graph. This allows efficient replacement of an origin that does not hold the minimal distance constraint and more importantly, significantly speeds up the following step of the algorithm. Additionally, if the graph is large and $k \ll |V|$, which is assumed to be the common case, it is possible to select origins in a distributed manner, and only when all k origins are selected, validate that min_dist holds, as the probability of randomly selecting near-by vertices is low, especially in power-law graphs which are the focus of this work.

3.3.2 Approximating minimal distance

Enforcing a minimal distance between cluster origins ensures that clusters will not overlap, which may result in mixture of their features. As show in Sect. 4, a large graph makes the random selection with minimal distance constraint very effective. As shown in Algorithm 3, GKM uses a simple heuristic to approximate this distance. First, it approximates the distance between the two most distant vertices (line 1–3). It uses approximation since finding the exact two most distant vertices requires all-pair distance calculation, which is practically infeasible in large-scale graphs. Then, it divides this distance by a constant f which is a function of the number of clusters k. Considering a d-dimensional space as in standard k-means, this constant is simply obtained by $f = \sqrt[d]{k}$. However, since GKM uses the true distances, and we apply it on power-law graphs, even $f = \sqrt[2]{k}$ is practically sufficient. The parameter $0 < \gamma \leq 1$ enables relaxing this distance in order to avoid excessive searches of the vertices that will be used as origins, as a result of a too large min_dist.

Algorithm 3 APPROXMINIMALDISTANCE

```
Require: Vertex set $V$, number of clusters $k$, distance function $d(v_i, v_j)$
Ensure: Cluster origin set $O$
1: $v_i \leftarrow Select\ v \in V\ uniformly\ at\ random$
2: $v_j \leftarrow \arg\max_w d(v_i, w)$
3: $v_k \leftarrow \arg\max_w d(v_j, w)$
4: Return $\gamma \cdot d(v_k, v_j)/f(k)$
```

3.3.3 Assigning vertices to clusters

Algorithm 4 shows the main phase of GKM. Once the origins are selected, GKM performs a single iteration over all the vertices and assigns each vertex to the cluster whose origin is the nearest. During this phase, it is possible to use the already calculated distances from each origin to all other vertices; therefore, no other distance calculations are required, and only k comparisons are performed for each vertex.

Considering only the distances between vertices may result in "mega-clusters" [17], i.e., extremely large clusters, which are the outcome of highly popular files that "pull" many other files to their cluster. These mega-clusters make the clustering of the graph significantly less useful since they do not provide a clear separation between the vertices, and additionally, they do not scale down the original problem, keeping it in a similar order of the number of vertices.

In order to balance the number of vertices between clusters, we define a size balancing threshold τ, which is used to determine whether a vertex can be shifted between clusters. When a vertex is "close" to several cluster origins, then it is assigned to the cluster that has the smallest number of vertices, and not necessarily the cluster that has the nearest origin. We define the set of clusters that a vertex is "close" to, as the clusters that have an origin which is at most τ farther than the nearest cluster origin, and select the smallest cluster from this group (line 6). This method relaxes the nearest distance optimization depending on the value of τ, where high values of τ result in a more relaxed distance optimization and a more balanced size of the obtained clusters.

Due to the random nature of the selection of cluster origin, it might be speculated that the resulting clusters may vary significantly between runs of GKM. However, the heuristic

Algorithm 4 AssignVerticesToClusters

```
Require: Vertex set $V$, Cluster origin set $O$, distance function $d(v_i, v_j)$, size balancing threshold $\tau$
Ensure: Cluster set $C$
1: for all $v \in V$ do
2:     $D^v \leftarrow \emptyset$
3:     for all $o \in O$ do
4:         $D^v[o] \leftarrow d(v, o)$
5:     end for
6:     $c^* \leftarrow \arg\min_{c \in \{|D^v[o_c] - \min D^v| \leq \tau\}} |D^v[o_c]|$
7:     $C_{c^*} \leftarrow C_{c^*} \cup v$
8: end for
9: Return $C$
```

behind the algorithm design is that due to the large size of the graph, its power-law nature (having a few very popular and strongly connected vertices), and the min_dist limitation, clusters quickly "grow" toward different popular vertices (i.e. popular files) and extend from there. These popular vertices are quite similar to the "attractors" that form the theory behind MCL. The empirical study on a real p2p network performed in Sect. 4 shows that indeed this is the case. Pathological cases can rise when extremely "remote" and not well-connected vertices are selected as cluster origins. In these unlikely cases, however, these clusters will be completely disconnected or include only a few vertices and hence can be either discarded or manually looked at in case they capture a very unique set of similar vertices.

3.3.4 Efficiency

The algorithm performs k times single-source shortest path (in case the minimal distance constraint holds for each random selection of cluster origin). We used a single-source Dijkstra [10], which can be efficiently calculated [4] in $\mathcal{O}(|E| + |V| \log |V|)$.

A simple improvement to the algorithm listed in Algorithm 1, during the selection of cluster origins, first calculate the distances between cluster origins, and only if a vertex sustains the minimal distance constraint, perform the remaining distance calculation using Dijkstra to all other vertices. This improvement is useful when there is a high probability to select cluster origins that are too close to each other. This can be a result of using a very large k relative to the size and density of the network, or running this algorithm on a network which is strongly connected with many high similarity edges, which is clearly not the case in p2p file similarity networks.

Once completed, the vertices are traversed once, mapping each to the cluster origin that matches the smallest distance. This phase can be completely avoided, if during the Dijkstra calculation from each cluster origin, only the nearest cluster and its corresponding distance is stored. Therefore, the overall time complexity of the improved algorithm is bounded by $\mathcal{O}(k(|E| + |V| \log |V|))$.

Another improvement is achieved by removing vertices from shortest path calculation in case they have shorter distance to a previous cluster origin. Consider a vertex v that have a distance d_j from cluster origin j, and distance $d_i < d_j$ from a previously calculated cluster origin i. Any path traversing through vertex v to another vertex will have shortest distance to cluster i than to cluster j. Therefore, it is possible to remove vertex v from shortest paths of cluster j.

Following the above observation, it is possible to obtain stronger bounds on the efficiency of the algorithm. Consider a graph that has all vertices aligned along a line with indices $1 \ldots n$, with a completely balanced set of clusters. Meaning, given n vertices and k clusters, each cluster holds exactly n/k vertices. In the worst case, cluster origins are aligned along the line graph, spaced with n/k vertices apart. The first cluster will have to traverse all $n-1$ vertices, the second $n-n/k-1$ (since only the first n/k are closer to the first cluster), the third $n-2n/k-1$, and so forth, until the kth cluster that will traverse only $n/k-1$ vertices. This linear descend causes the algorithm to traverse a $O(n^2)$ vertices.

The best case is when each cluster origin split the space of the previous cluster origin in half. In this case, the first cluster origin will be in the middle of the line graph (at index $n/2$) and will traverse all vertices. The second and third cluster origins will be at indices $n/4$ and $3n/4$ accordingly. Each of them will have to traverse only $n/2$ vertices. The following four clusters split each of the four regions and each traverses only $n/4$ vertices. Overall, the algorithm traverses $O\,(n\log(n))$ vertices.

While this analysis seems unfeasible when dealing with large graphs (due to their high dimensionality), it is, in fact, possible to find vertices that "split" the graph using the overall popularity of the files. A highly popular file is most likely "close" to every other file in the graph; hence, it makes sense to select it as the central vertex. Other less-popular files can also be selected as origins, enabling efficient partitioning of the space.

In terms of memory consumption, assuming B bytes (we used 4 bytes integer) are stored for identifying vertices, link weight (storing the normalized similarity), and distance between vertices. Therefore, the similarity graph can be efficiently stored using $B\cdot|V|+B\cdot|E|$ bytes. Storing the k origins requires additional $B\cdot k$ bytes. Assuming we use the previously mentioned improvements, for each vertex, we store the nearest origin and its corresponding distance, which requires additional $2B\cdot|V|$ bytes. Therefore, the total memory consumption is bounded by $\mathcal{O}(B\cdot(|V|+|E|+k))$.

The above analysis indicates that GKM is scalable. The performance of k-means is related to its convergence parameter; hence, it is expected to see similar behavior with slight advantage to k-means, as it is known to efficiently converge given good selection of parameters. MCL is significantly less scalable than the rest due to its large matrix multiplications. Metis and Graclus are both designed for efficient partitioning of the data hence are likely to run faster than all other algorithms.

3.3.5 Network dynamics

Peer-to-peer (P2P) networks are highly dynamic as content is constantly added and removed by users. Adding and deleting files to the network changes the similarity graph, hence may affect the resulting clusters. In most cases, when a set of clusters is already established, the objective is to quickly associate new content with existing clusters without affecting other files. In this case, for each file that is added to the graph, the distance to each of the k origins must be calculated, which can be achieved by traversing only vertices adjacent to the new vertex. The distance of each adjacent vertex is added to the link that connects it to the new vertex, and the minimal distance is the one that is used to map the vertex to the correct cluster.

However, new files that gain extreme popularity introduce new links with large similarity values and can lead to "short-cuts" in the graph, resulting in a need to move vertices between clusters. Moreover, but less likely, trends in the network can result in the creation of new clusters that represent a true change in user preferences. When this is the case, the algorithm must run again on the updated graph. A possible improvement was suggested [6] using an approximation method that help identify vertices that are likely to change clusters, therefore focus only on them.

4 Evaluation

Evaluation is performed using a graph that represents a snapshot of the music files that were shared in the Gnutella [35] p2p network. It is also possible to evaluate the proposed method on a synthetic power-law graph and validate our claims regarding bias in current algorithms. We use a large-scale power-law graph originated from a real-world network, since it provides additional domain-specific information such as metadata about the songs. This realistic data set helps to show how commonly used efficiency measures provide misleading information about the applicability of the resulting clusters for search and recommendation purposes.

4.1 Data set

The files from Gnutella were collected by a 24 h active crawl over the shared folders of over 1.2 million users on November 25, 2007, selecting only musical mp3 files. At the time of the crawl, Gnutella was considered as the most popular file sharing network [1]. Using this data, a songs similarity graph was created, having songs as vertices and the weight of a link connecting two vertices as the total number of different users that shared both files. The graph holds a total of 531k songs, performed by more than 100k artists covering over 3,600 genres and have more than 1 billion interconnecting links. The songs are indexed in descending order of their popularity; i.e., given songs $i < j$, song i is more popular (shared by more users) than song j.

Many of the links represent a very weak relation between songs due to invalid or scarce content and hence were filtered. During the collection of songs, we only kept links that appear in at least 16 different users (16 was selected for implementation reasons). This filter guarantees that "weak" ties between songs are not inserted to the similarity graph. A second filter keeps, for each file only the top 40 % links (ordered by descending similarity value) and not less than 10 links, i.e., keeping 20 million undirected links in the graph.

4.2 Crawl accuracy

Collecting the complete information about all users and files in a p2p network is practically not feasible. Therefore, in order to understand how well the crawl captures the underlying network, an empirical study of the utility of an exhaustive crawl relative to a partial crawl is performed. Partial crawl means that not all users are reached, hence not all songs are collected, and the weights of the links between songs are less accurate.

The first validation is achieved by creating a smaller data set using a random sample of 100k users. While the sample set constitutes for less than 10 % of the original users, the number of songs in the sampled set is over 96 % of the overall songs (511k out of 530k). This shows that most users in the p2p network share similar files indicating that it is not required to perform an exhaustive crawl in order to obtain sufficient representative data.

Notice, however, that the songs that are lost during the sampling belong to very specific musical niches. These are of less interest to this work since this music is usually known to the users that like it, and they usually know how to find it. However, some trendy songs that are not yet shared by many users can be overseen indicating that detecting "hypes" require a more exhaustive crawl of the network.

A second analysis is done on the similarity graph itself. We construct smaller sub-networks, by including, for each vertex, only the top N neighbors, ordered by nonincreasing normalized similarity. The truncated links stand for weak relations between songs that disappeared since users holding these songs were not crawled. The number of times each song appears as the

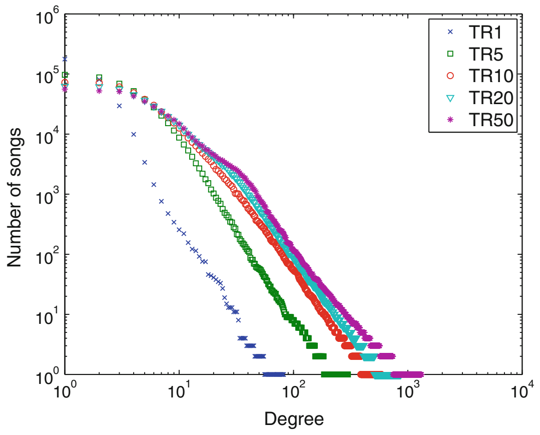


Fig. 1 Degree distribution of the song-similarity graph

nearest neighbor for increasing values of N was calculated. The results indicate that for $N \geq 10$ (we denote the graph $TR10$), the distributions almost overlap.

Finally, the degree distribution depicted in Fig. 1 shows, as expected, a power-law distribution for all samples, with 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.

4.3 Applying clustering algorithms

In order to evaluate the results of the different aspects of the considered algorithm, we ran all of them using the graph $TR10$ with $k = 100$. For GKM, the distance function that was used is given $d(i, j) = -\log_2(w(i, j))$. This function helps distinguish better between "near" and "far" songs, while mid-range is less affected. We note that we experimented with other distance functions, such as $1/\hat{w}(i, j)$ and $1 - \hat{w}(i, j)$. Although the results were very similar, it is likely that this is due to the similar smoothing effect of these functions over the similarity values, where it is likely that other distance methods might result in significantly different results [33]; hence, care should be taken when converting similarity to distance.

GKM longest running code is the Dijkstra calculations, which even in an efficient implementation take around 5 s on average for the evaluated graph. This results in an overall execution time of roughly 8 min, when the origins need to be reselected, and a few seconds when distances between origins and all other vertices are known (e.g., when changing various input parameters and recalculating).

Although Metis and Graclus operate on the similarity values, the published implementations require integer values. Hence, the similarity value was multiplied by 10,000 and then rounded to the nearest integer. This loses some precision since the normalized similarity value has longer fractions; however, multiplying by higher numbers resulted in a never-ending run time, which we attribute to implementation issues rather than the design of the algorithm. When the values are properly set, both algorithms run extremely fast and complete in roughly 20 s.

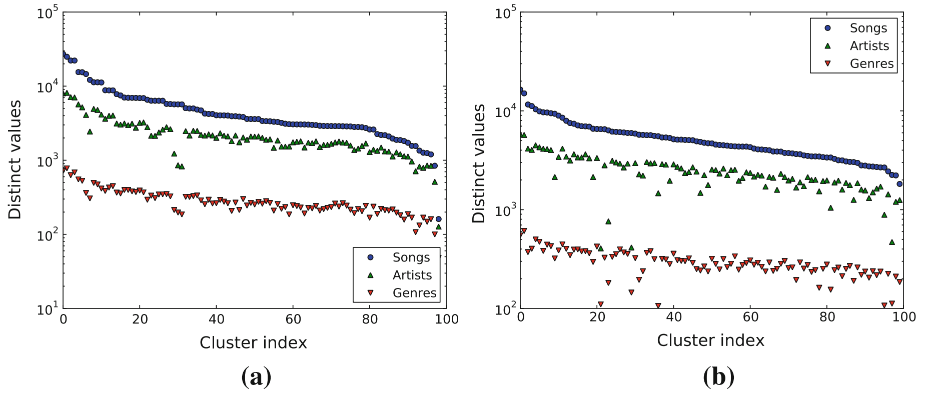


Fig. 2 Number of songs, distinct genres and distinct artists (**a** GkM, **b** k-means and Metis)

MCL does not accept as input the number of clusters k, but instead accept a value that specified the required coarseness of the resulting clusters. For this analysis, we tried several values; however, we found that even for the coarsest level, MCL produced 129k clusters and took 3 days to complete. Obviously, even without detailed comparison, this makes MCL really problematic for large networks. We note, however, that recently [38] several improvements to MCL were suggested without changing the underlying theoretical basis, making it somewhat more suited for such networks; hence, we include it in this analysis.

Since k-means does not run directly on graphs, the graph was first embedded into a 10-dimension feature matrix using fast sparse embedding [32]. This technique requires 10 shortest path calculation, which was efficiently calculated in less than a minute. Then, k-means was applied on the resulting matrix, with a stopping criteria of maximal centroid shift. Meaning, the algorithm converged once all centroids moved less than 0.5 units (measured in the Euclidian embedded space) relative to their prior location. This converges very fast and took less than a minute to complete.

4.4 Efficiency measures

Evaluating the efficiency of clusters can be very subjective [15] and highly dependent on the specific domain. Therefore, we use both conventional efficiency measures and develop additional domain-specific measures that utilize the metadata extracted from the shared files. Furthermore, some measures are used to study the robustness of GKM to various input parameters values.

4.4.1 Cluster size

The size of the cluster is defined as the number of different files it contains. Although we would like to obtain a uniform size distribution along the different clusters, such a distribution will not be correct for p2p shared content in general and musical content in particular. The reason is that usually there are some mainstreams and many much smaller niches (in our data set, there are more than 3,600 genres). In this case, the mainstream clusters are expected to be larger than the ones that represent a niche or some subgenres.

Looking at the number of songs in Fig. 2, GKM and k-means exhibit quite similar median number of songs, where, as expected, GKM exhibits more "noisy" clusters. This is attributed

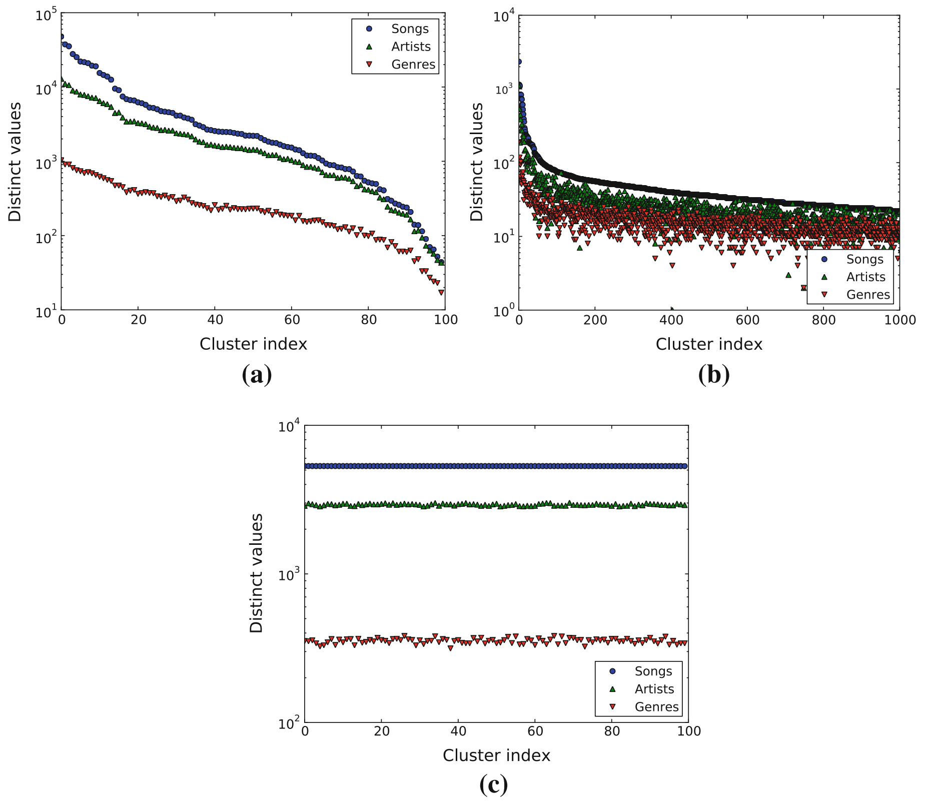


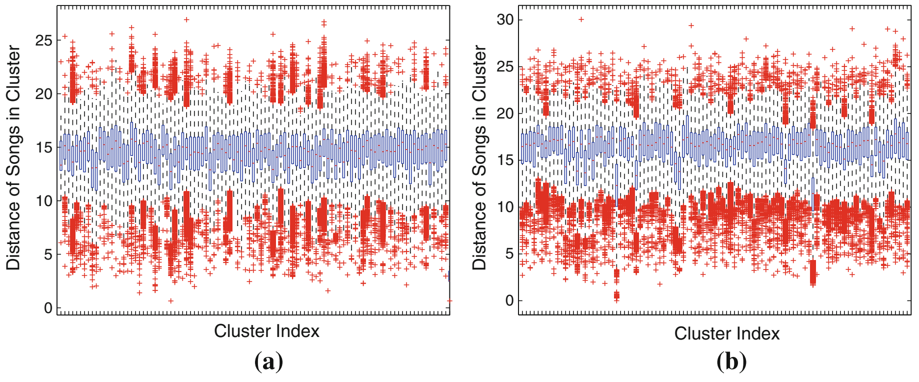
Fig. 3 Number of songs, distinct genres, and distinct artists, showing problematic results of **a** Graclus, **b** MCL, and **c** Metis

to the observation that GKM's clusters grow according to the distances of vertices, whereas k-means converges to a local minima which is an average of near-by vertices.

Figure 3 shows the results of Graclus, MCL and Metis, all of which exhibiting problematic behavior. Graclus exhibits a large variety of sizes, with a few extremely large clusters. The largest cluster contains almost 400k songs (not shown in the plot scale), which are 75 % of the songs in the graph, rendering the resulting clusters completely useless for improving search and recommendation applications. As expected, MCL fragments the output clusters into over 129k clusters, most of them are very small (only the first 1k are shown). While it is possible to use only the k largest clusters, the tail is so long that unless k is extremely large, most files will be lost.

As expected, Metis exhibits an almost perfectly balanced size of 5,318 files in each cluster, and interestingly, the genres and artists are also quite smooth. However, using a completely uniform distribution via the balls-in-bins formula provides similar results given that songs are grouped into batches of 10–12. This can be explained by strong relations between songs that are in the same album (each containing 10 songs on average), hence are commonly downloaded together.

GKM provides the best overall average songs to artists ratio (i.e., the number of songs in each cluster divided by the number of unique artists performing these songs, averaged over all clusters) of 0.49, compared to the second best, Metis, with 0.55. Furthermore, since



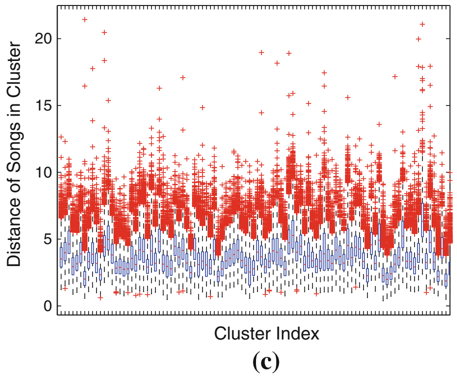


Fig. 4 Distance distribution and size balancing threshold τ. **a** GKM $\tau = 0$. **b** GKM $\tau = 3$. **c** k-means

GKM relaxes the size balancing constrain, it generates more natural clustering, with larger clusters for the more mainstream songs and smaller clusters for the less-popular songs. In the remainder of the paper, we mostly focus on the results of GKM and k-mean and compare to other methods only when the results are not of their problematic partitioning of the data set.

4.4.2 Distance distribution

Unlike k-means, GKM does not guarantee convergence of cluster distances to local minima. The reason is that GKM does not shift the cluster origins and its use of the size balancing parameter τ, which prefers balanced size distribution over distance minimization. Hence, it is important to empirically validate that the resulting clusters have a relatively uniform distribution of distances, with different values of τ.

Figure 4 depicts the distribution of the distances of all files in each cluster, using k-means and GKM with $\tau = 0$ (no size balancing) and with $\tau = 3$. The latter was chosen once we found that the median distance resulting from an execution without size balancing is 15. We empirically selected τ to be 20 % of this value, i.e., $\tau = 3$.

The central mark in each box of the boxplot is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to 1.5 times the difference between the 25th and the 75th percentile distances, and + sign beyond the whiskers is considered as outliers.

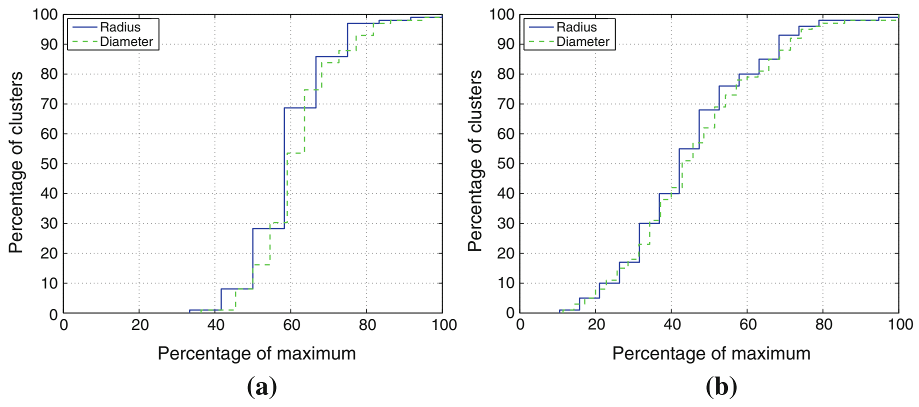


Fig. 5 Radius and diameter. **a** GKM. **b** k-means

The immediate observation is the k-means results in very "tight" clusters with a much smaller median distance than the GKM. Such a result is expected, when considering the k-means algorithm. Even the many outliers that exist in k-means extend mostly to smaller regions than GKM. However, GKM manages to create a uniform distribution of distances, with much less outliers ("bad" outliers being the high ones). As expected, when using higher value of τ, the median distance is slightly higher and the distances are less uniform, since the distance optimization is relaxed in favor of size balance. Notice that all figures include extreme outliers that potentially can be extracted into different clusters.

4.4.3 Radius and diameter

The radius of a graph is defined as the minimum over the shortest paths between all vertices, whereas the diameter is defined to be their maximum. While these measures have many implications in scale-free graphs [3,5], in our case, they reflect whether most clusters are relatively "tight" or are spread on a relatively large space, i.e., how similar are the songs contained in the cluster. In order to compare different algorithms, we normalize the radius and diameter by their maximal value, taken over all clusters. Note that the radius and diameter can be obtained from all algorithms, even those that do not define a cluster origin. Hence, these measures are in some sense more useful for comparing the different algorithms than the distance distribution.

Figure 5 compares the cumulative distributions function (CDF) of the radius and the diameter percentages of the GKM and k-means. GKM results in relatively spread clusters, having 80 % of the clusters within 50–70 % of the maximum cluster radius and diameter. This fact aligns with the previously analyzed distance distribution. k-means results in slightly tighter cluster, due to its optimization targeting procedure.

4.4.4 Normalized cut

Recall that the normalized cut of a cluster measures the ratio between links outgoing from a cluster to other clusters and the overall cluster degree. Often, good clustering procedures results in small cut value. We extend this measure to weighted graphs by defining a weighted normalized cut:

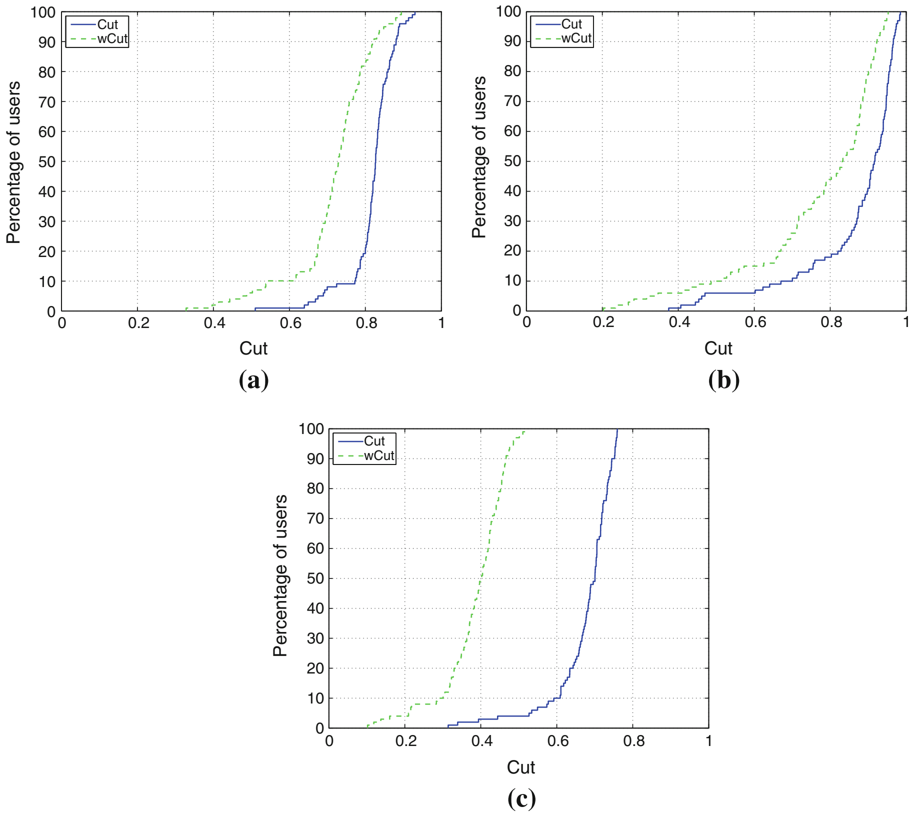


Fig. 6 Normalized cut and normalized weighted cut. **a** GKM. **b** k-means. **c** Metis

$$wNcut(C) = \frac{\sum_{i\in C, j\notin C} A(i,j)}{\sum_{i\in C}\sum_j A(i,j)} \tag{4}$$

Figure 6 plots the CDF of the normalized cut and normalized weighted cut obtained by GKM and k-means, compared with the near-optimal value of Metis (lower cut value is better). GKM performs better than k-means as a result of GKM shifting each vertex toward the cluster that have the highest sum of neighbor weights. While this procedure modification significantly improved the cut and weighted cut, the obtained result is still higher than Metis. This shows that the normalized cut is not always the best measure for successful clustering when considering the applicability of the resulting clusters.

4.4.5 Diversity of popularity

To be efficiently used by recommender systems, clusters are required to contain files with large diversity of popularity. Such a diversity allows the recommender system to recommend content which is less popular therefore often difficult to find [8]. However, it is expected that clusters that correspond to more mainstream content will have more popular files, while clusters that correspond to niches will have less-popular files.

Figure 7 shows the distribution of file popularity per cluster (lower song index indicates higher popularity). The two extreme cases are Metis that produces a very balanced set and

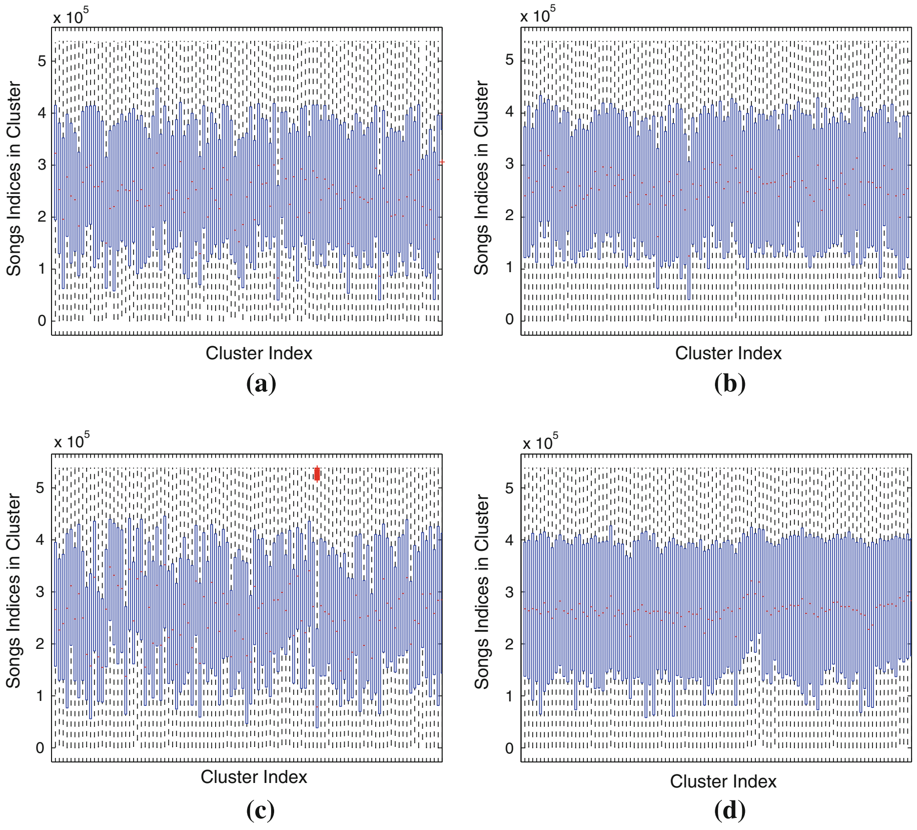


Fig. 7 Song popularity distribution. **a** GKM $\tau = 0$. **b** GKM $\tau = 3$. **c** k-means. **d** Metis

k-means that exhibits the least balanced distribution. The completely balanced clusters again indicate an almost uniform distribution of songs into clusters. On the other hand, clusters that hold only highly popular content and clusters with mostly not-popular content are also less useful for a recommender system, since common collaborative filtering techniques are based on like-minded users causing them to support popular taste stronger than unpopular one.

GKM exhibits a distribution between the two extreme cases. Different values of τ enable adjusting the results, with higher τ values result in more balanced popularity. This is attributed to shifting songs between clusters in contrast to their "natural" attraction to other songs, which breaks tight groups of songs with similar popularity into different clusters.

4.5 Discussion

Different efficiency measures capture different aspects of the resulting clusters. Using a graph extracted from a real p2p network enables more accurate evaluation of the results than using a synthetic graph. The most important observation is that popular measures, which are often used as optimization factors for state-of-the-art clustering algorithms, may lead to clusters that are nonoptimal for many applications and particularly for search and recommendation implementations that are the focus of this paper. Clusters that have low cut values tend to be large clusters, mainly due to the skewed nature of the network power-law graph. This fact causes some domain-specific measures to look good. Yet by examining the underlying reasons, one reveals that most algorithms will not perform well in real-world applications.

GKM appears to be well-suited algorithm for search and recommendation applications. Although it is not the fastest algorithm of the ones we have tested, its resulting clusters exhibit a good overall mixture of size and diversity, with tight distance and small set of music types contained in each cluster. Graclus, Metis, and MCL "suffer" from their optimization goals, leading them to produce too large clusters, clusters with strict groups of songs, or too many clusters. All of these outcomes make the clusters appear good with respect to some efficiency measures, but, as we have shown, to perform poorly in measures that are more likely to be of interest in real-world application.

Although k-means has similar optimization goals as the GKM algorithm, the former is more strict regarding minimal distance optimization. Overall, k-means distance approximation techniques perform quite well, and it has the advantage over GKM of balancing computation speed and accuracy, which is determined by the dimension of the embedded space. The only measure in which the k-means was found inferior to the GKM was the diversity of popularity: having some clusters with only less-popular songs while other clusters with only highly popular songs. The diversity of the popularity in clusters is required by recommender system and less for efficient content searching; hence, k-means can be efficiently used in such applications.

5 Use case: recommender system

As a measure for a real-world application, we used the clusters for creating a recommender system that provides users with songs that are related to the ones they already share. Since people are assumed to have a defined taste in music, we expect that given a good clustering scheme, the songs of each user will reside in a small number of clusters and a large fraction the user's songs in one of the clusters, which we refer to as the *dominant* cluster. By mapping each user to her dominant cluster or even to several clusters, we can potentially identify the musical taste of the user. 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 which is known to be hard [18].

First, we seek to evaluate how well the different clustering methods perform under these metrics. Using the original user-to-song network, we count, for each user, the number of clusters in which her songs reside. Within the dominant cluster (the one that holds most of the user's songs), we count the prevalence of the user's songs, i.e., the percentage of songs that are contained in the dominant cluster. We expect that a good clustering method does not spread users songs in too many clusters, and the dominant cluster of any given user should hold a significant portion of her songs.

Figure 8 plots the CDFs of the number of clusters and the prevalence in the dominant cluster for the clustering methods. The figure shows that GKM and Metis are similar (kk-means, which is not shown, resulted very similarly to GKM). For these, roughly 8 % of the users have all their songs in a single cluster, while 70 % of are mapped to less than 30 different clusters.

Graclus and MCL performed very differently. Graclus resulted in relatively low number of clusters and high prevalence (desired outcome), while MCL resulted in a very high number of clusters (we limit the x-axis to 100; however, the maximal number of clusters for one user was 675) and low prevalence. While these results are quite expected due to the way these

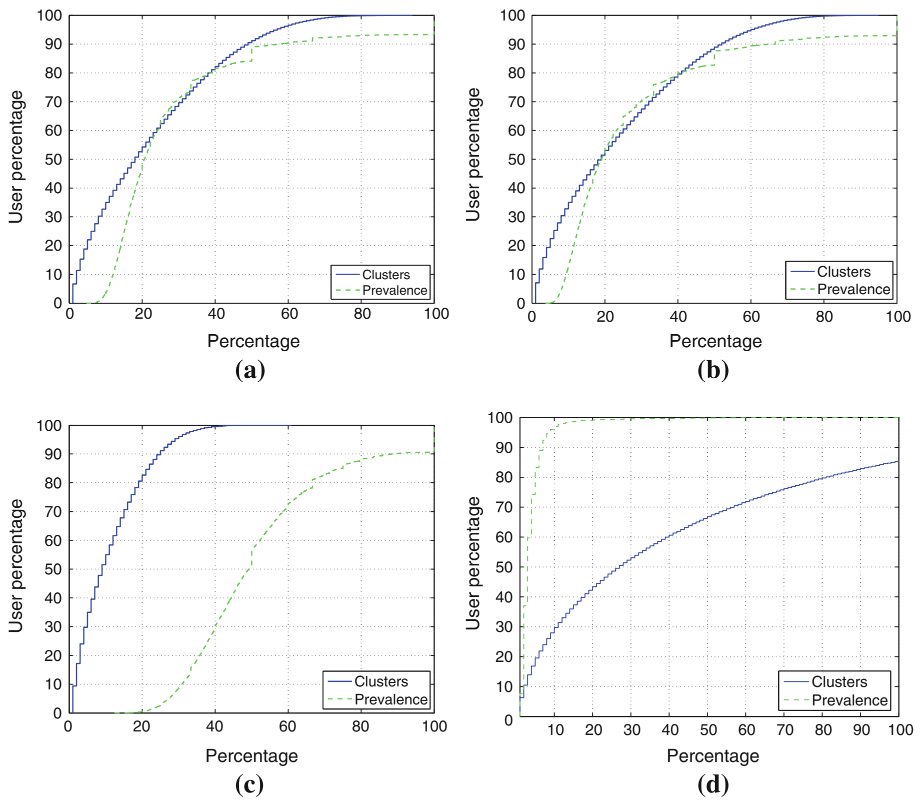


Fig. 8 Results of mapping users to clusters, showing the number of clusters that holds each user's songs and the song prevalence in each user's dominant cluster. **a** GKM. **b** Metis. **c** Graclus. **d** MCL

algorithms operate, they are only a single factor for the success of the recommender system, as we show next.

Our recommender system works as follows. We generate recommendations to a user by simply finding the user's dominant cluster (we break ties arbitrarily) and recommending the most popular songs from that cluster. This is a fairly naive approach, but it shows a simple use case for the clustering method by leveraging the properties of the clusters and can be executed extremely fast.

We compare the results of this method with two other methods, which we refer to as "most popular" and "nearest neighbor". The "most popular" method simply recommends the overall most popular songs that a user does not have. We consider this method as a lower bound for success, as it completely ignores the songs the user has and instead relies on the assumption that the popular songs are (by definition) liked by many users; thus, it is likely that they are a good recommendation. A major advantage of the "most popular" method and our proposed method is that preparing the list of possible recommendations can be preprocessed once for all users, and then the recommendations can be performed extremely fast.

The "nearest neighbor" method considers all the neighboring songs to the known user's songs and iteratively recommends the nearest neighboring song. We consider this method to be an "upper bound" for success since it uses all the information in the graph and it is significantly more costly than the other methods (obviously, this is not a real upper bound,

Table 1 Validation results using the "most popular" and "nearest neighbor" baseline methods, and using the popular songs within dominant clusters generated with the different clustering methods.

	Most popular (%)	Nearest neighbor (%)	GKM (%)	k-means (%)	Metis (%)	Graclus (%)	MCL (%)
Precision	22.7	38.4	27.3	30.8	17.0	22.5	36.9
Recall	7.0	10.1	8.6	9.1	5.8	7.2	9.3

but compared to the popularity method, we expect it to perform better). It is significantly more computationally expensive than the popularity-based method, as it requires per-user traversal of the song-similarity graph.

Evaluating the results is done by randomly selecting a training set of 60 % of the songs of each user. Then, each recommender system attempts to recommend the remaining 40 % of the songs (evaluation set) by using the above-mentioned methods. Since the likelihood of finding the exact missing user's songs is low, we measure success by looking at the artists. We count the precision, i.e., how many artists were correctly recommended out of the recommended set, and recall, i.e., how many artists were correctly recommended out of the real data. Clustering was performed on the TR20 graph, and we used $k = 100$ for all algorithms that accept it as input.

Table 1 summarizes the results of this analysis. The simplest "most popular" approach reaches a precision of 22.7 %, whereas our best "nearest neighbor" approach reaches a precision of 38.4 %. Surprisingly, MCL performed the best among the different clustering methods; however, since the number of clusters is huge and as Fig. 8d shows, users songs are spread among many clusters, finding the dominant cluster for each user is a costly task, similar in the scale to the best method. Putting computational complexity aside, the high precision obtained by MCL is unexpected since the prevalence of the dominant cluster is very low compared to the other methods, showing the MCL manages to place a small majority of the user's songs in a cluster which is potentially interesting to the user.

The k-means slightly outperforms GKM (30.8 and 27.3 %, respectively), mainly because, as we have shown, the way the popularity is distributed in GKM may cause our algorithm to recommend less-popular songs that are more difficult for the user to find; however, these are harder to evaluate. Despite Metis apparently having similar spread of users' songs (as can be seen in Fig. 8b), it performed the worst. This again shows that the hard constrain Metis imposes on the distribution of songs has real implications on its applicability in practice. The recall of all methods is quite low, showing that, overall, is it quite difficult to find the exact artists in the evaluation set.

Although these results seem somewhat low, they are relatively high when considering the vast amounts of songs that exist and shared by p2p users. For example, one user in the data set had songs tagged with "Bob Dylan ft. Van Morrison", "Chuck Berry" and "Bob Dylan". Given only two songs which were both tagged with "Bob Dylan ft. Van Morrison", the "nearest neighbor" algorithm recommended songs that were tagged with "Van Morrison". Although this is clearly a good recommendation, it was tagged as a failure by our automatic evaluation procedure.

This analysis shows that even when using extremely sparse, noisy, and implicit ranking, it is still possible, yet not trivial, to recommend the songs and artists that a user may prefer. Moreover, devising an automatic criteria for accurate estimation of the correctness of recommendations is a nontrivial task. The popularity-based methods were found to be quite successful due to the fact that many users prefer popular mainstream music. However, the proposed “nearest neighbor” method can be further refined so that it provides diversity in the recommendation. For example, recommending the song that has smallest average distance to all other songs, or for more adventurous users, recommending the nearest neighboring song that is the least popular one. These recommendations, however, are much more subjective and thus are difficult to assess a priori.

Finally, we note that we evaluated the robustness of GKM to the selection of the number of clusters (k). When discussing clusters of musical taste, the value of k is closely coupled with the number of various musical “flavors” that are captured in the underlying graph. We found that for $k \geq 50$, the results are similar to the above, showing that the number of clusters that are actually needed is finite, which is important given that the time and space complexity of GKM are strongly tied with the value of k. Even when the number of clusters is high, users are mapped to strongly dominant clusters; thus, the results are not significantly differ than what we observed for $k = 100$.

6 Conclusion

This paper aims to provide a profound understanding on the applicability of clustering p2p networks, focusing on *“search and recommendation”* implementations. The observation that many p2p networks, as well as other well-used networks such as the WWW and various social services, exhibit a power-law degree distribution raises the concern that state-of-the-art clustering algorithms produce biased clusters. Furthermore, commonly used efficiency measures often overlook *search and recommendation* applicability issues and provide wrong impression that the resulting clusters are well suited for these domains.

In order to overcome these issues, we propose the GKM, a clustering algorithm that operates on large-scale power-law graphs and implements a relaxed distance optimization procedure with size balancing. Using a real-world power-law graph that provides us with domain-specific information, we perform a comparative analysis and show that distance-optimizing techniques, such as GKM and the approximated k-means, result in clusters that are well suited for *search and recommendation* applications.

References

1. Ars technica report on P2P file sharing client market share. http://arstechnica.com/old/content/2008/04/study-bittorren-sees-big-growth-1
2. Anglade A, Tiemann M, Vignoli F (2007) Virtual communities for creating shared music channels. In: Proceedings of international symposium on music information retrieval
3. Barabási A-L, Albert R (1999) Emergence of scaling in random networks. Science 286:509–512
4. Barbehenn M (1998) A note on the complexity of Dijkstra’s algorithm for graphs with weighted vertices. IEEE Trans Comput 47(2):263
5. Bollobas B, Riordan O (2004) The diameter of a scale-free random graph. Combinatorica 24(1):5–34
6. Bradley PS, Fayyad U, Reina C (1998) Scaling clustering algorithms to large databases. Knowl Discov Data Min (AAAI Press)
7. Bradley PS, Fayyad UM (1998) Refining initial points for k-means clustering. In: ICML ’98. Morgan Kaufmann, San Francisco (pp. 91–99)
8. Celma O, Cano P (2008) 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
9. Dhillon IS, Guan Y, Kulis B (2007) Weighted graph cuts without eigenvectors a multilevel approach. IEEE Trans Pattern Anal Mach Intell 29(11):1944–1957

10. Dijkstra EW (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1:269–271
11. Dongen SV (2000) Performance criteria for graph clustering and markov cluster experiments. Technical report. National Research Institute for Mathematics and Computer Science
12. Faloutsos C, Lin K-I (1995) Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: ACM SIGMOD '95
13. Fessant FL, Kermarrec AM, Massoulie L (2004) Clustering in peer-to-peer file sharing workloads. In: IPTPS
14. Fodor I (2002) A survey of dimension reduction techniques. Technical report. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
15. Geleijnse G, Schedl M, Knees P (2007) The quest for ground truth in musical artist tagging in the social web era. In: ISMIR, Vienna
16. Gish AS, Shavitt Y, Tankel T (2007) Geographical statistics and characteristics of p2p query strings. In: IPTPS
17. Handcock MS, Raftery AE, Tantrum JM (2007) Model-based clustering for social networks. J R Stat Soc Ser A 170(2):301–354
18. Herlocker JL, Konstan JA, Terveen LG (2004) Evaluating collaborative filtering recommender systems. ACM Trans Inf Syst 22:5–53
19. Hu T, Sung S (2006) Finding centroid clusterings with entropy-based criteria. Knowl Inf Syst 10:505–514
20. Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv 31(3):264–323
21. Jin R, Goswami A, Agrawal G (2006) Fast and exact out-of-core and distributed k-means clustering. Knowl Inf Syst 10(1):17–40
22. Kang U, Tsourakakis C, Faloutsos C (2011) PEGASUS: mining peta-scale graphs. Knowl Inf Syst 27(2):303–325
23. Karypis G, Kumar V (1995) A fast and high quality multilevel scheme for partitioning irregular graphs. In: International conference on parallel processing
24. Koenigstein N, Shavitt Y, Weinsberg E, Weinsberg U (2010) On the applicability of peer-to-peer data in music information retrieval research. In: ISMIR
25. Luo P, Xiong H, Lü K, Shi Z (2007) Distributed classification in peer-to-peer networks. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '07. ACM
26. Mowat A, Schmidt R, Schumacher M, Constantinescu I (2008) Extending peer-to-peer networks for approximate search. In: 23rd annual ACM symposium on applied computing
27. Mowat A, Schmidt R, Schumacherand M, Constantinescu I (2008) Extending peer-to-peer networks for approximate search. In: ACM SAC '08. ACM, New York. pp 455–459
28. Narasimhamurthy A, Greene D, Hurley NJ, Cunningham P (2010) Partitioning large networks without breaking communities. Knowl Inf Syst 25(2):345–369
29. Navarro G (2001) A guided tour to approximate string matching. ACM Comput Surv 33:2001
30. Ars technica report on P2P file sharing client market share. http://arstechnica.com/old/content/2008/04/study-bittorren-sees-big-growth-1
31. Pelleg D (2000) Moore A X-means: extending k-means with efficient estimation of the number of clusters. In: The 17th international conference on machine learning. Morgan Kaufmann, Los Altos. pp 727–734
32. Platt JC (2004) Fast embedding of sparse music similarity graphs. In: Advances in neural information processing systems
33. Priness I, Maimon O, Ben-Gal I (2007) Evaluation of gene-expression clustering via mutual information distance measure. BMC Bioinform 8(1):111–123
34. Resnick P, Varian HR (1997) Recommender systems. Commun ACM 40(3):56–58
35. Ripeanu M (2001) Peer-to-peer architecture case study: Gnutella network. In: First international conference on peer-to-peer computing
36. Sakuma J, Kobayashi S (2010) Large-scale k-means clustering with user-centric privacy-preservation. Knowl Inf Syst 25(2):253–279
37. Saroiu S, Gummadi KP, Gribble SD (2003) Measuring and analyzing the characteristics of napster and gnutella hosts
38. Satuluri V, Parthasarathy S (2009) Scalable graph clustering using stochastic flows: applications to community discovery. In: KDD
39. Scholkopf B, Smola A, Muller K-R (1998) Nonlinear component analysis as a kernel eigenvalue problem. Neural Comput 10(5):1299–1319
40. Shavitt Y, Weinsberg E, Weinsberg U (2010) Estimating peer similarity using distance of shared files. In: International workshop on peer-to-peer systems (IPTPS)

41. Shavitt Y, Weinsberg E, Weinsberg U (2011) Mining music from large-scale peer-to-peer networks. IEEE Multimedia 18(1):14–23
42. Shavitt Y, Weinsberg U (2009) Song clustering using peer-to-peer co-occurrences. In: adMIRe
43. Sripanidkulchai K, Maggs B, Zhang H (2003) Efficient content location using interest-based locality in peer-to-peer systems. In: INFOCOM
44. Steinbach M, Karypis G, Kumar V (2000) A comparison of document clustering techniques. In: KDD
45. Stutzbach D, Rejaie R (2006) On unbiased sampling for unstructured peer-to-peer networks. In: ACM IMC, pp 27–40
46. Stutzbach D, Rejaie R, Sen S (2007) Characterizing unstructured overlay topologies in modern P2P file-sharing systems. In: Internet measurement conference (IMC), pp 49–62
47. Voulgaris S, Kermarrec A-M, Massoulié L, van Steen M (2004) Exploiting semantic proximity in peer-to-peer content searching. In: 10th international workshop on future trends in distributed computing systems (FTDCS 2004), China
48. Wang F, Li P, König AC, Wan M (2012) Improving clustering by learning a bi-stochastic data similarity matrix. Knowl Inf Syst 32(2):351–382
49. Wong B, Vigfússon Y, Sirer EG (2007) Hyperspaces for object clustering and approximate matching in peer-to-peer overlays. In: USENIX HOTOS ’07. USENIX, Berkeley, pp 1–6
50. Wu J, Xiong H, Chen J (2009) Adapting the right measures for k-means clustering. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, KDD ’09. ACM, New york
51. Yang B, Garcia-Molina H (2002) Improving search in peer-to-peer networks. In: ICDCS ’02: proceedings of the 22nd international conference on distributed computing systems
52. Zaharia MA, Chandel A, Saroiu S, Keshav S (2007) Finding content in file-sharing networks when you can’t even spell. In: IPTPS
53. Zheng R, Provost F, Ghose A (2007) Social network collaborative filtering. In: 6th workshop on ebusiness (WEB)

Author Biographies

Irad Ben-Gal is the Head of the IE program in the Department of Industrial Engineering at Tel Aviv University. His research interests include statistical methods for control and analysis of stochastic processes; applications of information theory and machine learning to industrial and service systems. He holds a B.Sc. (1992) degree from Tel Aviv University, M.Sc. (1996) and Ph.D. (1998) degrees from Boston University. He wrote and edited five books, published more than 80 scientific papers, patents, and book chapters, and has received several best papers awards. He is a member of the Institute for Operations Research and Management Sciences (INFORMS), the Institute of Industrial Engineers (IIE), The European Network for Business and Industrial Statistics (ENBIS), and an elected member in the International Statistical Institute (ISI). He is a department editor in the IIE Transactions on Quality and Reliability and serves in the Editorial Boards of several other professional journals. Prof. Ben-Gal supervised dozens of graduate students and received several research grants, among them from General Motors, IEEE, the Israeli Ministry of Science, and the European Community. He worked with such companies as Pratt & Whitney, Siemens, Proctor and Gamble, Kimberly-Clark, Applied Materials, IBM and more.

Yuval Shavitt received his B.Sc., M.Sc., and D.Sc. degrees from the Technion, Haifa, Israel, in 1986, 1992, and 1996, respectively. After one year as a postdoctoral fellow at Johns Hopkins University at Baltimore, MD, he joined the networking center at Bell Labs, Lucent Technologies, Holmdel, NJ, where he worked on network management, QoS routing, and wireless networks. Since 2000, he is a faculty at the School of Electrical Engineering at Tel Aviv University. In 2004, he incepted the DIMES project for mapping the Internet infrastructure, which revolutionized the field of Internet measurement. In addition to his work on network measurements and analysis, Prof. Shavitt works on algorithms for the analysis of large graphs and on information retrieval.

Ela Weinsberg received her M.Sc (2011) from the department of industrial engineering at Tel Aviv University, Israel, and her B.Sc (2006) in computer science and mathematics from Bar-Ilan University. Her research interests include data mining and analysis of large-scale networks.

Udi Weinsberg is a researcher at the Technicolor research lab in Palo Alto, CA. Udi received his PhD (2011) and M.Sc (2007) from the school of electrical engineering at Tel Aviv University, Israel, and his B.Sc (2001) from the Technion, Haifa, Israel. His research focuses on the intersection of user profiling, privacy, and recommender systems. In addition, he studies large-scale networks, the topology of the Internet, and peer-to-peer networks.