The Times They Are a-Changin':
Evolving Communities in a Musician Network

Cristian Zanoci and Jim Andress

December 11, 2016

1 Introduction

Network analysis has become an important tool for understanding the structure and evolution of large interacting systems. Although most of the research in the field is centered around on-line social networks, recently there has been a lot of progress in applying network analysis techniques to other fields, such as finance, retail and academia. For this project we study the Million Song Dataset (MSD) [1], which contains information about artists, songs, and the relationships between them over the past 90 years. When viewed collectively, it can be difficult to gain any insight from the more than 44,000 artists in the dataset. By using the tools and algorithms of network analysis, though, it is possible to divide musicians into smaller communities within the larger music scene. These smaller communities are more easily understood and can help researches extract information from the network.

However, the extensive time span of MSD provides some unique challenges for network analysis techniques. In particular, the links between artists, as well as the artists themselves, come and go, meaning that the usual community detection algorithms designed for static networks can miss the bigger picture of these dynamic processes. Luckily, there have recently been several attempts to design evolutionary clustering algorithms that help retrieve information about the temporal evolution of a network [2, 3, 4]. In our project, we seek to investigate these algorithms and explore how the clusters they produce differ from those produced by algorithms intended for static networks. Accurately tracking the changes in these communities over time is important as it can better reveal long-term phenomena that occur in the musical network and potentially even predict the future behavior of musician communities.

The rest of this paper is organized as follows. In Section 2 we review the classic Clauset-Newman-Moore algorithm for community detection in static networks, along with more recent methods in evolutionary community detection on dynamic graphs. Section 3 describes the data collection process, the methods we use for community detection, and our metrics for evaluating their performance. In Section 4 we present the results of applying these algorithms to the community detection problem in the musicians' network and discuss their implications.

2 Literature Review

As described earlier, community detection within a static network is a common problem which has been extensively studied. The techniques used to tackle this problem vary widely, from standard clustering methods to spectral matrix decomposition algorithms. Of these, one of the most widely used is Clauset et al.'s efficient algorithm for identifying sub-communities within a network [5]. Specifically, Clauset et al. define the concept of modularity, which takes

a partition of the network's nodes into communities and measures its goodness, depending on the fraction of edges in which both endpoints lie in the same cluster. Intuitively, partitions with high modularity have strong ties within and weak ties between communities. After defining this metric, the paper goes on to describe an efficient algorithm which places nodes into communities by greedily maximizing the modularity at every step. A nice property of their approach is that the number of communities does not need to be chosen ahead of time; rather, the algorithm simply terminates once the modularity begins decreasing, leaving an ideal number of communities.

Although CNM is at this point a classic algorithm, it is only intended for use on static networks. Thus, if we want to fully utilize the temporal dimension of our data, we must turn to relatively new research into evolving community detection. The first attempt to study the behavior of communities as a function of time came from Hopcroft et al. [6]. They performed community detection via hierarchical clustering at different snapshots in time and tried to find stable communities in a citations graph. However, this approach was less successful than those to come since they didn't use information about the communities at earlier times in their detection of communities at later times.

In 2006, Chakrabarti et al. [7] introduced the notion of temporal smoothness where the cluster membership at time t is influenced by the clusters at an earlier time $t-1$. The idea was to maximize a weighted sum which consists of the snapshot quality - a measure of accuracy of the partition given the graph structure at time t - and the history cost, which describes how well the partitions at times $t-1$ and t fit together. Later, Chi et al. [4] modified the spectral clustering algorithm to take into account temporal smoothness. They used graph cut as a metric for measuring community structures and evolutions. Unfortunately, since the algorithm requires multiple eigenvector computations, it tends to be slow and scales poorly with network size.

One of the most successful evolutionary clustering methods is FacetNet by Lin et al. [2]. They start with a cost function similar to Chakrabarti's, which consists of a snapshot cost and a temporal cost. These costs are defined in terms of the KL divergences between community structures at two different times. Then they provide an iterative algorithm which minimizes the objective cost function. The framework is flexible enough to allow for different number of nodes and clusters at times $t-1$ and t, which was not possible in the previous algorithms.

Finally, Kim and Han [3] proposed a particle-and-density based evolutionary clustering method. They introduced the concept of a nano-community, which is their basic unit of a temporal network that spans across multiple years. They also proposed a cost-embedding technique that allows for temporal smoothing by pre-processing the data, which is independent of the clustering algorithm. The key advantage of this method is its flexibility and an order of magnitude runtime improvement with respect to FacetNet.

3 Methods

3.1 Data Collection

For our underlying network, we are using the Million Song Dataset (MSD). This publicly available data was released in 2011 and includes songs from 1922 on. Each entry contains a song's name, artist, year of release, musical features, and (most importantly for us) similar

artists. By creating nodes for each artist and treating the MSD similar artists as edges, we are able to construct a musician network linking similar artists within and across years.

Although it is undeniable that a musician's influence can persist for decades after they last put out a song, we are interested in modelling the state of the music scene at particular moments in time. Therefore, we are using a simple model of network evolution, in which artists are added to the graph the first year that they appear in the MSD, edges are present between artists only if they are both in the network, and artists are removed from the graph five years after their last occurrence in the MSD. By evolving the network in this way, we are able to measure the similarity of artists as they were perceived during the years they were active. This evolution process results in a different musician network for every year from 1930 to 2010. The number of artists per year ranges from only 50 in the 1930's to over 18000 in the 2000s.

Actually retrieving the data was one of the challenges we faced at the start of our project. The dataset is extremely large (roughly 300GB), but we are only interested in a small portion related to the artists themselves, not the tracks. We walked through each of the files on the AWS server hosting the data to get the relevant portions stored in a SQL database, and even this simple filtering took almost 20hrs. When we were finished, we were left with about 700MB of artist data.

3.2 Detecting Evolving Communities

Our main focus for this project is to compare the communities discovered in the musician network using static algorithms and those intended for dynamic graphs. The three algorithms we will be focusing on are described in some detail below.

3.2.1 Clauset-Newman-Moore (CNM)

The CNM algorithm is a well known community detection algorithm for static graphs. It works by greedily optimizing the *modularity* of a community partition. If a graph with m edges is partitioned into communities such that c_v represents the community containing node v, then modularity is defined as

$$Q = \frac{1}{2m} \sum_{vw} \left[A_{vw} - \frac{k_v k_w}{2m} \right] \delta(c_v, c_w) \tag{1}$$

where A_{vw} indicates whether there is an edge between nodes v and w, k_v represents the degree of node v, and δ is the Kronecker delta function. Intuitively, the modularity measures how much the proportion of within-community edges deviates from the expected value in a random graph. Thus, high modularity indicates the presence of strongly connected communities which are relatively unconnected from the rest of the network.

The CNM algorithm works by first assigning each node to its own individual community. Then, the algorithm iteratively merges the pair of clusters which leads to the largest increase in modularity. At some point during this greedy process we will find that merging any of the remaining clusters actually decreases the modularity, at which point the algorithm terminates. Thus, the number of clusters does not need to be passed as a parameter to the algorithm.

3.2.2 FacetNet Evolutionary Clustering

As mentioned in the literature review, Lin et al.'s implementation [2] minimizes a cost function

$$cost = \alpha \cdot CS + (1 - \alpha) \cdot CT, \tag{2}$$

where the snapshot cost CS measures how well a community structure fits the graph at time t, and the temporal cost CT measures the consistency between the current community at time t and the previous community at time $t-1$. We expect that the community structure shouldn't change dramatically within one time step.

At any time t, we represent the information about the current graph G_t on n nodes through its adjacency matrix $W \in \mathbb{R}^{n \times n}$, normalized such that $\sum_{i,j} W_{ij} = 1$. The main idea is that the currently observed graph structure W can be viewed as a combined outcome of the interaction between m communities. Let $\Lambda_{kk} = p_k$ be the prior probability that the entry W_{ij} is due to the kth community and let $X_{ik} = p_{ik}$ be the probability that an interaction in community k involves node i, where $X \in \mathbb{R}^{n \times m}$ and $\Lambda \in \mathbb{R}^{m \times m}$. Since these are probabilities, they are normalized as $\sum_{i=1}^{n} X_{ik} = 1$ and $\sum_{k=1}^{m} \Lambda_{kk} = 1$. Then Lin et al. show that $X\Lambda X^T$ is the marginal distribution induced by the community structure at time t and well approximates W, i.e. $W_{ij} \approx \sum_{k=1}^{m} p_k \cdot p_{ik} \cdot p_{jk}$.

In order to define the snapshot cost CS, we use the Kullback-Leibler divergence which tells us how closely does the community structure $X\Lambda X^T$ approximate the true data W:

$$CS = KL(W || X\Lambda X^T). \tag{3}$$

Similarly, define the temporal cost to be the KL-divergence between the community structure $Z = X_{t-1}\Lambda_{t-1}X_{t-1}^T$ at time $t-1$ and the one at time t, which tells us how dramatically the clusters change over time:

$$CT = KL(Z || X\Lambda X^T). \tag{4}$$

The authors of [2] show that we can use an iterative approach to minimize the *cost* in Eq. 2 by performing the following updates:

$$X_{ik} \leftarrow X_{ik} \cdot \sum_{j=1}^{n} \frac{(\alpha W_{ij} + (1-\alpha) Z_{ij}) \cdot \Lambda_{kk} \cdot X_{jk}}{(X\Lambda X^T)_{ij}} \tag{5}$$

$$\Lambda_{kk} \leftarrow \Lambda_{kk} \cdot \sum_{i,j=1}^{n} \frac{(\alpha W_{ij} + (1-\alpha) Z_{ij}) \cdot X_{ik} \cdot X_{jk}}{(X\Lambda X^T)_{ij}} \tag{6}$$

At the end of the update, we have to normalize back the distributions: $\sum_{i=1}^{n} X_{ik} = 1$ and $\sum_{k=1}^{m} \Lambda_{kk} = 1$. Note that the most expensive step in the algorithm is computing the matrix products $X\Lambda X^T$.

It is worth noting that in order to treat the case when there is a different number of nodes in the network at times $t-1$ and t, we have to adjust the size of Z from $n_{t-1} \times n_{t-1}$ to $n \times n$. We accomplish this by removing the rows and columns of Z which correspond to nodes that don't exist anymore at time t, and adding rows and columns of zeros for every node that didn't exist at time $t-1$ but is present at time t.

In this framework, an artist can participate in multiple communities at the same time and with different participation levels. If we define the diagonal matrix $D \in \mathbb{R}^{n \times n}$ with entries $D_{ii} = \sum_{j=1}^{m} (X\Lambda)_{ij}$, then $(D^{-1}X\Lambda)_{ij}$ represents the participation level of artist i in community j. Even though FacetNet uses soft community membership, we impose the restriction that in the end a node should belong to only one cluster, namely the one in which it has the highest participation level. This is done to make FacetNet consistent with the CNM and DBSCAN, which both impose hard community memberships.

Lastly, we show how to determine the optimal number of clusters m. Extend the concept of modularity from Eq. 1 to handle soft membership by defining the soft modularity:

$$Q_s = Tr[(D^{-1}X\Lambda)^T(D^{-1}X\Lambda)] - \vec{1}^T W^T (D^{-1}X\Lambda)(D^{-1}X\Lambda)^T W\vec{1}, \tag{7}$$

where $\vec{1}$ is a vector of all ones. To detect the best number of communities at time t, we run our algorithm for a variety of m and pick the one that maximizes the soft modularity Q_s.

3.2.3 Density-Based Evolutionary Clustering with DBSCAN

Unlike other evolving community detection algorithms which try to achieve temporal smoothness by manipulating the final communities, Kim and Han's density-based approach instead focuses on smoothing the networks provided as input to the community detection algorithm. If we let $N(v)$ represent all nodes directly connected to node v, then the *structural similarity* metric $\sigma(v, w)$ given by

$$\sigma(v, w) = \frac{|N(v) \cap N(w)|}{\sqrt{|N(v)| \cdot |N(w)|}} \tag{8}$$

roughly represents the proportion of neighbors which v and w share. This similarity metric can be used to create a weighted version of a graph, where the weight of the edge between any two nodes is simply their structural similarity. Kim and Han then build an adjacency matrix whose weights linearly interpolate between those of the network in time $t-1$ and time t. Because the weights incorporate multiple years' worth of data, the resulting adjacency matrix ought to vary more smoothly across the decades.

In order to actually extract the communities from the weighted adjacency matrix, we follow Kim and Han's suggestion of running the DBSCAN algorithm, which is a standard density-based clustering algorithm. DBSCAN works by identifying the core nodes, which are any nodes that have at least η nodes within a distance of ϵ from it. Then, two nodes a and b are in the same cluster if there is a chain $a = p_0, p_1, \ldots, p_{k-1}, p_k = b$ such that $p_1, \ldots, p_{k-1}$ are all core points and $d(p_i, p_{i+1}) \leq \epsilon$.

Based on this definition, nodes which are far from all the others will not be put into clusters by the standard DBSCAN algorithm. However, in order to be consistent with the other algorithms we are using, we need all nodes to be partitioned. We therefore added a post-processing stage in the DBSCAN algorithm which partitioned the remaining nodes. First, we run the original algorithm and obtain all the clusters. Then, we find the node which has the most neighbors contained in one cluster, and we add the node to the cluster. We then repeat this process, including the newly labelled node in the counts for all other nodes. This modification to DBSCAN allows the algorithm to efficiently find the cores of the clusters, at which point we add in the outlier nodes along the cluster edges.

3.3 Evaluating Algorithm Success

As we have hopefully made clear, the detection of evolving communities in a network is a question of balancing multiple competing interests. Although we want the communities to accurately reflect the data at any given point in time, we also recognize that there are global trends through the lifetime of the network which impose extra constraints on the smoothness of the communities and how they change over time. In order to track the performance of algorithms along each of these competing dimensions, we carefully chose different metrics with which to test our results.

3.3.1 Snapshot Cost: Modularity

The first way in which we can evaluate an evolving community detection algorithm is through its snapshot costs. The snapshot cost of a specific year is simply a measure of how well the communities detected in that year reflect the network in that year. Because each year is viewed in isolation, any standard community detection metric can be applied for the snapshot costs. We therefore chose to use the modularity of the communities to quantify their snapshot costs as this metric is widely used and intimately connected to the CNM algorithm.

3.3.2 Temporal Cost: Rand Index

The other fundamental cost associated with evolving communities is the temporal cost, which measures how much the communities change over time. We chose the Rand Index as a metric for temporal costs as it is a simple and intuitive translation of the desired criteria that the communities in time t and $t+1$ are similar. Suppose that $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$ represent the networks at times t and $t+1$. We consider all nodes present in both networks $V = V(\mathcal{G}_t) \cap V(\mathcal{G}_{t+1})$. For any pair of nodes $v, w \in V$, we either have that the nodes v and w have the same community state in times t and $t+1$ (i.e. they are either in the same community in both times or different communities in both times) or they have different community states. If s represents the number of pairs of nodes in V which have the same community state and d represents the number of pairs of nodes in V with different community states, then the Rand Index between times t and $t+1$ is

$$R_{t,t+1} = \frac{s}{s+d} \tag{9}$$

The Rand Index has a value of 1 if the communities are identical in two time steps, a value of 0 if a single large community splits into individual disconnected nodes, and a value of roughly 0.5 if two equally sized communities merge.

3.3.3 Community Explanation: Artist Term Similarity

While the two metrics just described do quantify important structural properties of a network partitioning into communities, one of the goals of a community detection algorithm is to provide insight which is of value to humans. Thus, the communities produced from such an algorithm are of particular interest when they coincide with some natural intuition about the data. In order to show that the communities we are investigating are consistent with human evaluation of the data, we make use of the MSD's *artist terms* field. Each artist in the dataset has a set of associated terms describing their act, their music, and their time period. Let $T(v)$

represent the set of terms associated with artist v. We can then compute the Jaccard similarity between the terms for two artists as

$$J(v, w) = \frac{|T(v) \cap T(w)|}{|T(v) \cup T(w)|} \tag{10}$$

We expect that on average, pairs of vertices which are put into the same cluster ought to have a higher Jaccard similarity than pairs of vertices put into different clusters. As our metric we use the ratio of average Jaccard similarity for pairs of nodes belonging to the same community over the average Jaccard similarity for pairs of nodes belonging to different communities.

4 Results

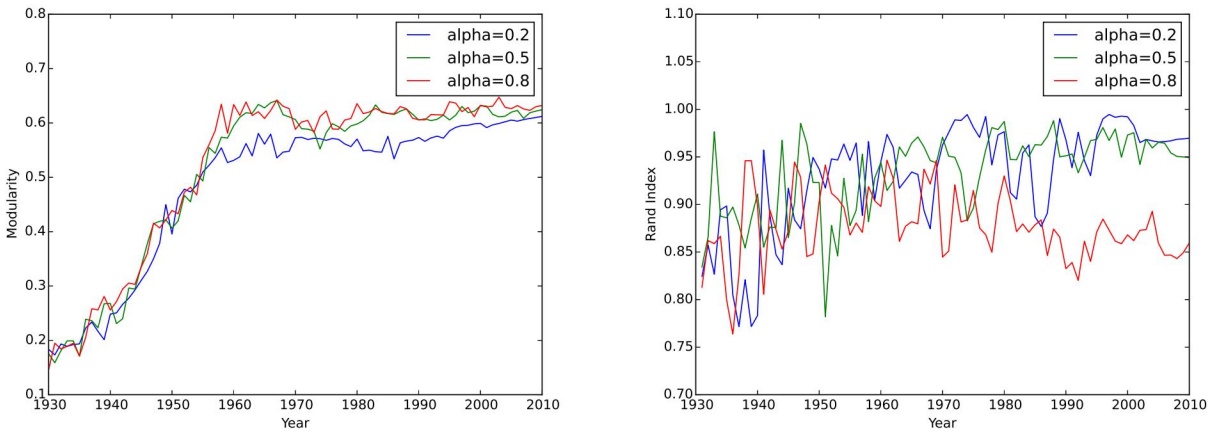
4.1 Choosing Algorithm Parameters

A nice feature of the CNM algorithm is that it has no parameters which need to be chosen manually by the user; however, this is unfortunately not the case for DBSCAN or FacetNet. The DBSCAN algorithm takes as input the parameters η, the minimum number of points to be a core node, and ϵ, the maximum search radius. In order to choose appropriate values for these parameters, we ran a simple grid search over different combinations of parameters and recorded the maximum cluster size, the percentage of nodes which weren't put into a cluster, and the total number of clusters detected at each year. We then chose the set of parameters which did the best job of minimizing these three values simultaneously.

This minimization is quite challenging, as these three features are all intimately connected. For instance, because the clusters in DBSCAN expand greedily, for most sets of parameters we either detected a single large cluster (high maximum cluster size, low percentage not clustered, low number of clusters) or hundreds of small, separated clusters (low maximum cluster size, high percentage not clustered, high number of clusters). In the end, we settled on $\eta = 4$ and $\epsilon = 0.5$, which led to the largest cluster containing about a third of the nodes (similar to the results in CNM), about a third of the nodes not put into a community until the post-processing step, and an average of less than 50 communities detected per year.

For FacetNet, we need to select the coefficient α, the range of values for the number of clusters, and the number of updates for X_{ij} and Λ_{kk}. For determining α, we looked at different trade-offs between CS and CT. Figure 1 show the modularity and temporal smoothness achieved by FacetNet for three values of α. These results coincide with our expectations: higher α optimize for snapshot cost and hence result in higher modularity, while lower α optimize for temporal smoothness, resulting in higher values of rand index. Based on these plots, we see that a values of $\alpha = 0.5$ provides a nice trade-off, achieving both high modularity and temporal smoothness.

In practice, we know that the number of clusters shouldn't change dramatically from time $t-1$ to t. This is supported both by our desire to minimize the CT term in the cost function, and by the observation that the number of music genre doesn't change much from year to year. Therefore, when searching for the optimal value of m, we explore the range $[m_{t-1}-3, m_{t-1}+3]$, where m_{t-1} is the number of clusters at time $t-1$. As for the number of iterations, we found that in most of the cases, the algorithm converges within 20 iterations.



(**a**) (**b**)

Figure 1: **(a)** The modularity and **(b)** the Rand Index for three different α values in FacetNet from 1930 to 2010.

4.2 Comparing CNM, DBSCAN, and FacetNet

Figure 2 shows how the three metrics, as well as the number of detected communities, changes from 1930 to 2010 when each of the three algorithms is run. As is clearly shown in subfigure (a), the DBSCAN algorithm produces significantly more clusters than the other techniques for the reasons described above. The results in subfigures (b), (c), and (d) are of much higher interest, though, as these show the trade offs between using algorithms designed for static networks (CNM) and those designed for dynamic networks (DBSCAN and FacetNet). In particular, (b) shows that while the modularity of the communities produced by DBSCAN is not significantly higher than CNM, the communities produced by FacetNet do have higher modularity (Wilcoxon signed-rank test $W = 1309, p = 0.10$ and $W = 418, p < 0.001$, respectively). Furthermore, (c) shows that the Rand Indices of both DBSCAN and FacetNet are higher than those of CNM ($W = 491, p < 0.001$ and $W = 775, p < 0.001$, respectively). Finally, (d) shows that the term similarity ratio for DBSCAN is higher than CNM's while FacetNet's does not differ significantly ($W = 451, p < 0.001$ and $W = 1305, p = 0.09$, respectively).

These results perfectly demonstrate the need for specialized algorithms while extracting evolving communities from networks. We have shown that these algorithms can produce communities which divide the network as well as the communities found by standard techniques (comparable modularities) and are as consistent with human intuition about the data (comparable term similarity ratios) but are significantly smoother in the time domain (higher Rand Indices). Because these resulting communities better encapsulate the time dependence of the data, they can be of more use to researchers interested in gaining insight into the macroscopic trends of the network, rather than individual trends during particular moments in time.

5 Division of Labor

- Cris Zanoci: Crawling and retrieval of original dataset, Implementation of graph building code, Implementation of CNM-based clusters, Implementation of FacetNet clusters

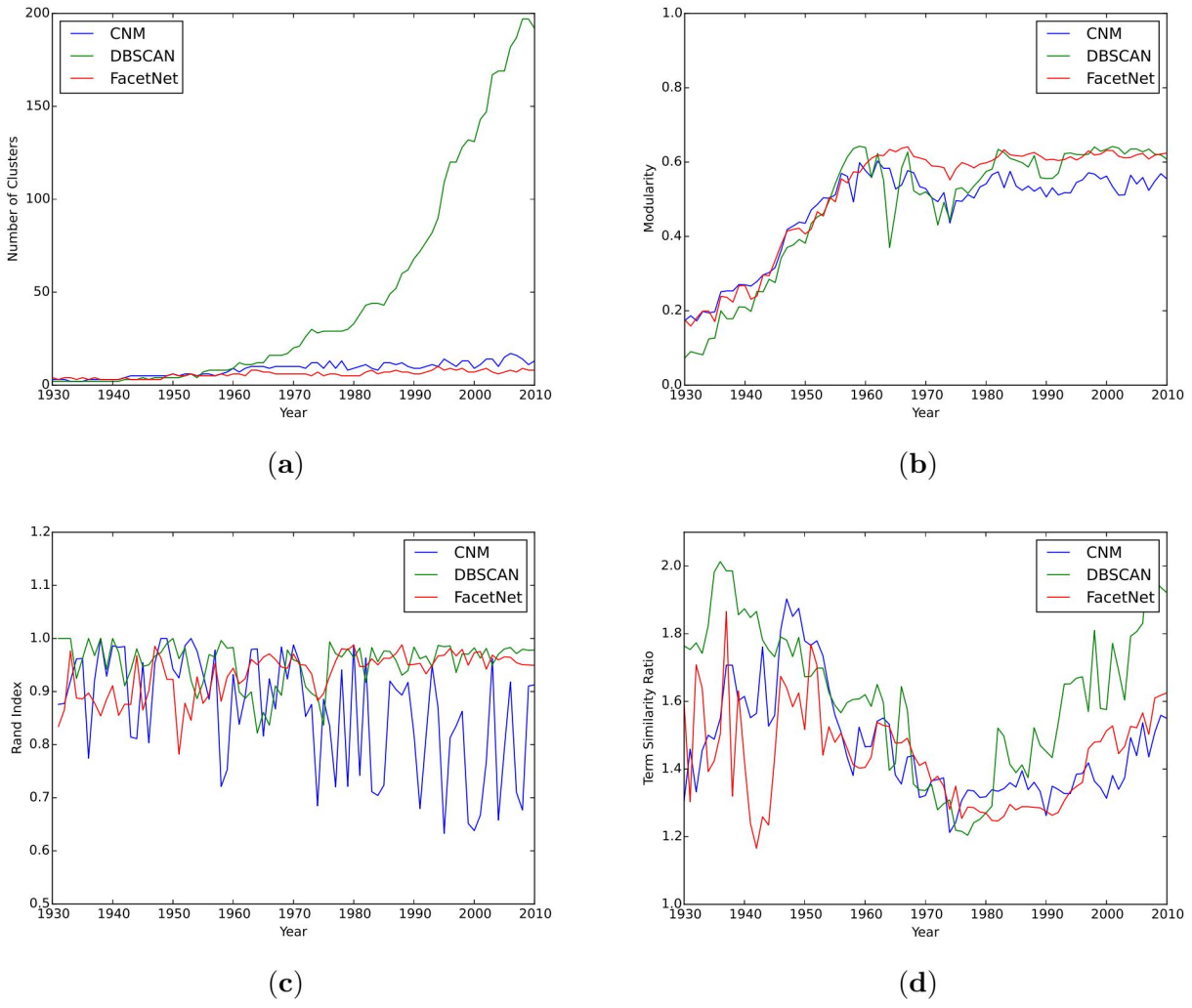


Figure 2: **(a)** The number of communities detected, **(b)** the modularity, **(c)** the Rand Index, and **(d)** the term similarity ratio for the three algorithm from 1930 to 2010.

- Jim Andress: Crawling and retrieval of original dataset, Implementation of database utilities, Implementation of metrics, Implementation of DBSCAN-based evolving clusters

We think that we should both receive the same grade.

References

[1] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *ISMIR*, vol. 2, p. 10, 2011.

[2] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, "Facetnet: a framework for analyzing communities and their evolutions in dynamic networks," in *Proceedings of the 17th international conference on World Wide Web*, pp. 685–694, ACM, 2008.

[3] M.-S. Kim and J. Han, "A particle-and-density based evolutionary clustering method for dynamic networks," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 622–633, 2009.

[4] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, "Evolutionary spectral clustering by incorporating temporal smoothness," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 153–162, ACM, 2007.

[5] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[6] J. Hopcroft, O. Khan, B. Kulis, and B. Selman, "Tracking evolving communities in large linked networks," *Proceedings of the National Academy of Sciences*, vol. 101, no. suppl 1, pp. 5249–5253, 2004.

[7] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 554–560, ACM, 2006.