PLOS | ONE

RESEARCH ARTICLE

Netgram: Visualizing Communities in Evolving Networks

Raghvendra Mall*, Rocco Langone, Johan A. K. Suykens

KU Leuven, ESAT-STADIUS, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium

* raghvendra.mall@esat.kuleuven.be

OPEN ACCESS

Citation: Mall R, Langone R, Suykens JAK (2015) Netgram: Visualizing Communities in Evolving Networks. PLoS ONE 10(9): e0137502. doi:10.1371/journal.pone.0137502

Editor: Renaud Lambiotte, University of Namur, BELGIUM

Received: October 31, 2014

Accepted: August 18, 2015

Published: September 10, 2015

Data Availability Statement: All relevant data are within the paper and its Supporting Information files. The Netgram tool is accessible in the MATLAB environment available here: "http://www.esat.kuleuven.be/stadius/ADB/mall/downloads/Netgram_Tool.zip".

Funding: This work was supported by EU: ERC AdG A-DATADRIVE-B (290923), Research Council KUL: GOA/10/-/09 MaNet, CoE PFV/10/002 (OPTEC), BIL12/11T; PhD/Postdoc grants-Flemish Government; FWO: projects: G.0377.12 (Structured systems), G.088114N (Tensor based data similarity); PhD/Postdoc grants; IWT: projects: SBO POM (100031); PhD/Postdoc grants; iMinds Medical

Abstract

Real-world complex networks are dynamic in nature and change over time. The change is usually observed in the interactions within the network over time. Complex networks exhibit community like structures. A key feature of the dynamics of complex networks is the evolution of communities over time. Several methods have been proposed to detect and track the evolution of these groups over time. However, there is no generic tool which visualizes all the aspects of group evolution in dynamic networks including birth, death, splitting, merging, expansion, shrinkage and continuation of groups. In this paper, we propose Netgram: a tool for visualizing evolution of communities in time-evolving graphs. Netgram maintains evolution of communities over 2 consecutive time-stamps in tables which are used to create a query database using the sql outer-join operation. It uses a line-based visualization technique which adheres to certain design principles and aesthetic guidelines. Netgram uses a greedy solution to order the initial community information provided by the evolutionary clustering technique such that we have fewer line cross-overs in the visualization. This makes it easier to track the progress of individual communities in time evolving graphs. Netgram is a generic toolkit which can be used with any evolutionary community detection algorithm as illustrated in our experiments. We use Netgram for visualization of topic evolution in the NIPS conference over a period of 11 years and observe the emergence and merging of several disciplines in the field of information processing systems.

Introduction

Large scale complex networks are ubiquitous in the modern era. Their presence spans a wide range of domains including social networks [1], biological networks [2], collaboration networks [3], trust networks [4] and communication networks [5]. These complex networks have a natural temporal aspect. Social networks evolve over time with addition and deletion of members, formation of friendships between people in different social circles or disappearance of friendship between people over time. In collaboration networks, a group of researchers working on a particular topic might collaborate intensively if they are working on an emerging topic whereas a group of researchers working together on an outdated topic might completely disappear over time.

Information Technologies SBO 2014-Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO, Dynamical systems, control and optimization, 2012-2017). Johan Suykens received the funding. ERC (European Research Council): http://erc.europa.eu/. FWO (Fonds Wetenschappelijk Onderzoek): http://www.fwo.be/en/. KU Leuven: http://www.kuleuven.be/English. BELSPO (Belgian Federal Science Policy): https://www.belspo.be/belspo/index_en.stm. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

Complex networks can be represented as graphs $G = (N, E)$ where N represent the vertices or nodes and E represents the edges or interaction between these nodes in the network. Most real-life networks exhibit community like structure i.e. nodes within a community are more densely connected to each other and sparsely connected to nodes outside that cluster. Traditionally, community detection methods [5–17] have focused on identifying communities in static representations of graphs.

However, by representing a time-evolving graph as a static network [10] it becomes difficult to detect and track the intrinsic changes in the community structures over time. By performing community detection on static snapshots [10] of the dynamic network, we loose the property of temporal smoothness which is essential to capture the evolution of communities over time. Temporal smoothness [18–20] allows to preserve the long-term trend in the dynamic graph while smoothing out short-term variations due to noise. This property is similar to the property of moving averages in time-series analysis [20]. In recent years, the problem of evolutionary community detection and its tracking in large scale dynamic social networks has received a lot of attention [19–29]. The goal of evolutionary community detection is to identify and track communities at different snapshots of time in non-stationary graphs. Throughout this paper we use the notation T to denote total number of time-stamps and t to denote an arbitrary time-stamp $t \leq$ T. We use C_j^t which comprises a set of nodes to represent the j^{th} cluster at time-stamp t and C^t (which is union of all C_j^t) to represent the set consisting of all the clusters C_j^t at time-stamp t.

In this paper, we propose a new tool Netgram which allows to visualize the evolution of communities in dynamic graphs. In order to visualize the progress of communities over time, Netgram follows a series of steps. Netgram takes as input the information about the individual identifier for all the nodes i.e. the number/label with which that node is represented in the network and its corresponding cluster membership at different time-stamps. It can be used as a post-processing step to any evolutionary community detection algorithm. After obtaining the input, in order to capture the significant events namely birth, death, merge, split, growth, shrinkage and continuation of communities, Netgram uses a modified version of the tracking algorithm proposed in [29]. We provide a visualization of the weighted network (W^t) at time-stamp t generated as a result of the tracking procedure. Once the evolution of communities is captured between two successive time-stamps, it is stored in a table. We then perform sql [30] join operations on these sets of tables to construct a query database. This query database contains information about evolution of all the communities over the different time-stamps. We then visualize this query database using separate colors for cluster identifiers and lines which track the evolution of individual communities. Netgram tries to adhere to certain design principles and a set aesthetic guidelines including minimizing the line cross-overs between the evolving communities during events like merge and split making it easier to track the evolution of individual communities in the dynamic network. The problem of minimizing the cross-talk between communities during different time-stamps is combinatorial by nature and Netgram uses a greedy solution for the same. Fig 1 illustrates the steps undertaken by the Netgram toolkit for visualizing evolution of communities. Fig 2 showcases the result that we get from the Netgram toolkit on a synthetic Birthdeath dataset of 1,000 nodes over 5 time-stamps generated from the software https://github.com/derekgreene/dynamic-community.

Related Work

We briefly mention here some of the methods that have been used in the past for detecting and tracking changes in complex networks. In [27] GraphScope was introduced. Graphscope is an efficient adaptive mining tool in time evolving graph for detecting communities. It requires no

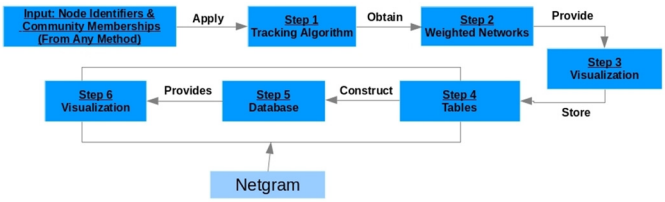


Fig 1. Steps undertaken by Netgram for visualizing evolution of communities in dynamic networks.

doi:10.1371/journal.pone.0137502.g001

user-defined parameter and operates completely in a standalone mode using the principle of Minimum Description Length (MDL) from information theory. Moreover, it can automatically detect communities and determine good *change-points* over time. The intuition is that communities do not change much over 2 consecutive time-stamps and thus have similar description lengths. This allows to group them together into a time segment in order to achieve better compression. Whenever a new snapshot cannot fit well into the old segment (in terms of compression), it introduces a change-point and starts a new segment at that time-stamp. These change-points detect drastic discontinuities in time.

In [25], the authors provided a framework called FaceNet. In this technique, the authors deviated from the traditional two-step approach to analyze community evolutions. In the traditional approach, communities are first detected for each time-stamp and then compared to determine correspondences. In this approach, the authors used a framework called FaceNet for analyzing communities and their evolutions through a robust unified process. This framework allowed the discovery of communities and captured their evolution with temporal smoothness [19] given by the historic community structures. The authors formulated their problem in terms of maximum a posteriori (MAP) estimation, where the community structure was estimated both by the observed networked data and by the prior distribution given by historic community structures. Then an iterative algorithm was developed which guaranteed convergence to an optimal solution.

In both the aforementioned techniques namely Graphscope [27] and FaceNet [25], the primary focus was on detection of communities in time evolving graphs rather than tracking the evolution of communities. These techniques can identify significant events like birth, death and continuation of communities. However, it is difficult to detect significant events like merge and split using these techniques.

In [24], a framework was provided which used the clique percolation method (CPM) for tracking evolution of communities in successive time-stamps. However, this technique is prone to be affected by noisy events. For example, if very few nodes from a community at one time-stamp split then this method detects it as a split event whereas these nodes might have been removed due to random fluctuations in community detection technique. In [22], a model was introduced which tracked the progress of communities over time in a dynamic network such that each community is characterized by a series of significant evolutionary events. By only keeping track of significant evolutionary events they overcome noisy events. However, none of these methods [22, 24] provide a visualization tool to observe and get insight into the evolution of communities in dynamic networks.

Recently, a method which maps changes in networks using alluvial diagrams was proposed in [21]. The method uses bootstrap sampling accompanied by significance clustering in order to distinguish meaningful structural changes from random fluctuations. This technique is

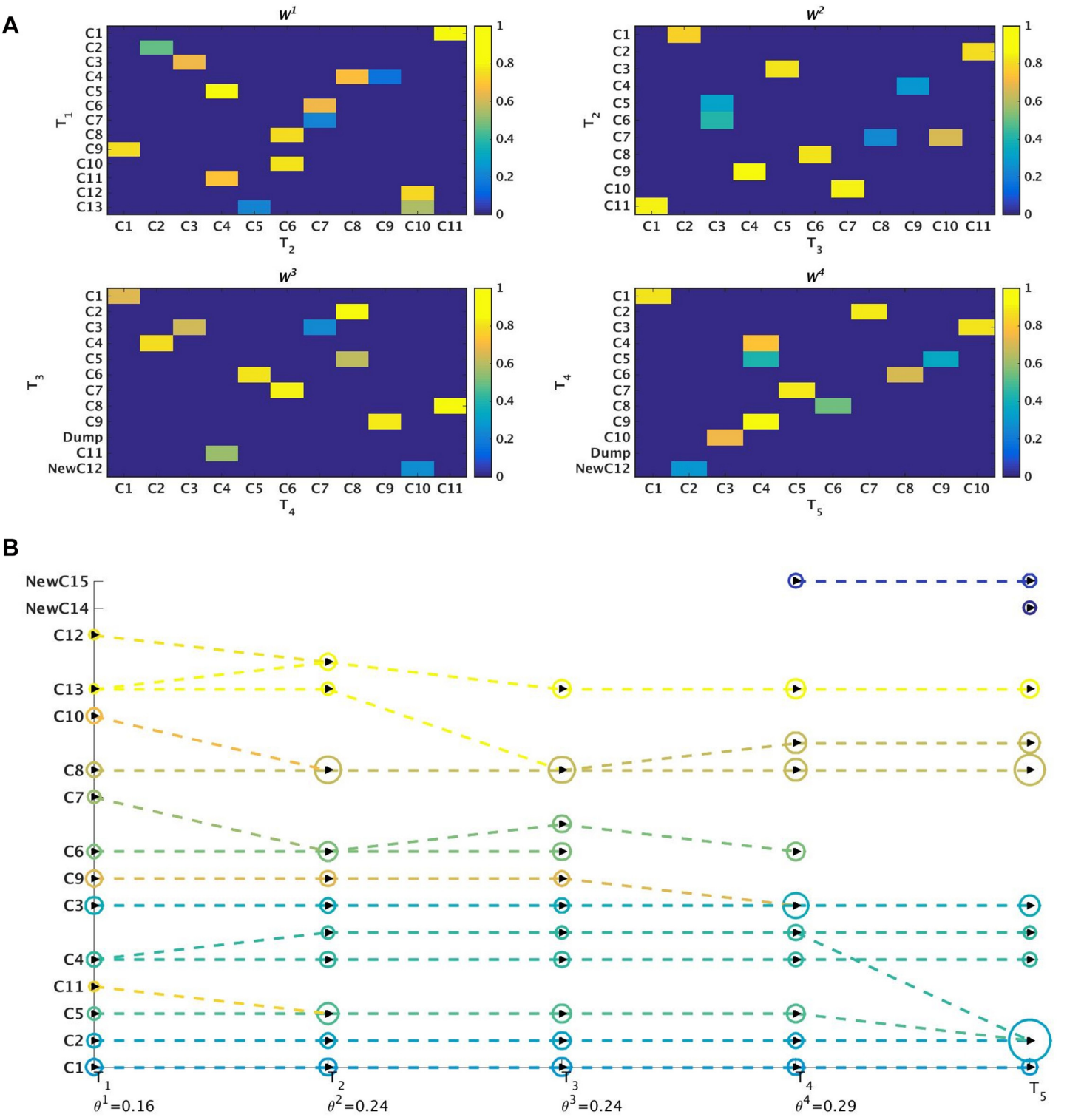


Fig 2. a) Visualization of the weighted networks (W^t) mapping evolution of communities over 5 time-stamps. W^t tracks evolution of a cluster between two consecutive time-stamps. The colors represent the weight of the edges in W^t. The weights can take value in the range [0, 1] i.e. $0 \leq w(V^t(j,k)) \leq 1$. b) Visualization and tracking of community evolution by Netgram for the clusters obtained from the Kernel Spectral Clustering with Memory Effect (MKSC) algorithm [28] for Birthdeath dataset. Netgram showcases the birth, death, merge, split, expansion, shrinkage and continuation of communities for the Birthdeath dataset over 5 time-stamps (T_1, T_2, T_3, T_4 and T_5). We represent each community with a different colour circle and the size is $\propto$ the number of nodes in that community. From Fig 2b, we can observe the death of clusters **C6** & **C7** at time-stamps T_3 and T_4 respectively. Similarly, we

can see birth of clusters **NewC14** & **NewC15** at time-stamp T_4 and T_5 respectively. We also observe that cluster **C11** merges with **C5** at time-stamp T_2 and cluster **C8** splits into 2 clusters at time-stamp T_4. We can observe that cluster **C8** has expanded at time-stamp T_3. The cluster **C8** also contracts at time-stamp T_4 as it splits into 2 clusters. Cluster **C1** demonstrates continuation over time.

doi:10.1371/journal.pone.0137502.g002

based on the principle introduced in [22]. However, this method [21] doesn't take into consideration the property of temporal smoothness [19]. The alluvial diagrams help to visualize events like merge, split, continuation, expansion and shrinkage of communities. However, they cannot showcase the birth and death of individual communities separately. This is because in this method [21] a new community can only emerge at a given time-stamp t as a split of some previous community at time-stamp $t-1$ and it is difficult to detect the death/dissolution of a particular community at one given time-stamp t. Moreover, this visualization using surfaces uses a large portion of the screen and makes it difficult to track the evolution of individual communities when there is a series of merge and split events. Netgram allows to separately identify birth of a new community, highlight the death of a community and makes it easy to identify and track individual community evolution in presence of multiple merge and split events using a simple line-based visualization system. The tracking algorithm of Netgram is based on similar principles as that proposed in [21, 22, 29] i.e. trying to identify significant events and differentiate it from random noisy events. However, the main goal of Netgram is to come up with a simple line-based visualization tool which allows to track evolution of individual communities, capture and visualize significant events like birth, death, merge, split, continuation, shrinkage and growth of communities in dynamic time-evolving graphs.

Evolution of Communities

Significant events that happen during evolution of communities in a dynamic network can be defined as:

- *Birth*: The emergence of a new community C^t_{new} at time t comprising nodes which were previously unseen at time $t-1$ i.e. $C^t_{new} \cap C^{t-1}_i = \emptyset$ with all the clusters C^{t-1}_i in the set $\mathcal{C}^{t-1}$ at time $t-1$.
- *Death*: The disappearance of one or more communities at time t. It suggests that the community which was appearing as C^{t-1}_i at a previous time has disappeared now i.e. $\frac{|C^{t-1}_i \cap C^t_j|}{|C^{t-1}_i \cup C^t_j|} < \theta^{t-1}$ for all the communities C^t_j in the set $\mathcal{C}^t$ at time t.
- *Continuation*: A community C^{t-1}_i remains intact for the next time t i.e $\frac{|C^{t-1}_i \cap C^t_j|}{|C^{t-1}_i \cup C^t_j|} \geq \theta^{t-1}$ holds true for a single community C^t_j at time t and it holds true in both time-directions. This distinguishes a continuation event from a merge event. Since, the community structure of C^{t-1}_i doesn't change much and continues to remain a single community C^t_j at time t, it is referred as a continuation.
- *Merging*: When the majority of the nodes from 2 or more communities at time $t-1$ for example C^{t-1}_h and C^{t-1}_i combine together to form one cluster C^t_j at time t. i.e. $\frac{|C^{t-1}_h \cap C^t_j|}{|C^{t-1}_h \cup C^t_j|} \geq \theta^{t-1}$ and $\frac{|C^{t-1}_i \cap C^t_j|}{|C^{t-1}_i \cup C^t_j|} \geq \theta^{t-1}$, all such conditions hold true. These clusters subsequently start to share a common time-line starting from time t.

- *Splitting*: When the majority of the nodes from a community C_i^{t-1} splits into 2 or more communities at time t say C_j^t and C_k^t i.e. $\frac{|C_i^{t-1} \cap C_j^t|}{|C_i^{t-1} \cup C_j^t|} \geq \theta^{t-1}$ and $\frac{|C_i^{t-1} \cap C_k^t|}{|C_i^{t-1} \cup C_k^t|} \geq \theta^{t-1}$, all such conditions hold true.
- *Growth*: When the number of nodes in the community C_i^{t-1} at time $t-1$ increase significantly for the corresponding community C_j^t at time t. For example, if the size of the community C_i^{t-1} increases by 20% at time t.
- *Shrinkage*: When the number of nodes in the community C_i^{t-1} at the time $t-1$ decreases significantly for the corresponding community C_j^t at time t. For example, if the size of the community C_i^{t-1} decreases by 20% at time t.
  The parameter θ^{t-1} is defined for each time-stamp $t \leq$ T. It helps to define the concept of majority or helps to keep track of events which are significant and prevents random fluctuations from being detected as merge or split events. It takes value between the interval [0, 1] and is explained in more detail in the next subsection.

Tracking Algorithm & Weighted Networks

In order to recognize these events we need a tracking algorithm that matches the communities found by the evolutionary clustering algorithms at each time step. Several such tracking algorithms have been developed including [22, 24, 29]. In this paper, we use a modified version of the tracking algorithm introduced in [29].

We first generate a weighted directed bipartite network W^t from the clusters at two consecutive time-stamps t and $t+1$. In case of a total of T time-stamps, we generate a set $\mathcal{W} = \{W^1, \ldots, W^{T-1}\}$ of weighted directed bipartite networks. Each bipartite network W^t creates a map between the set of clusters at time-stamp t i.e. $\mathcal{C}^t$ and time-stamp $t+1$ i.e. $\mathcal{C}^{t+1}$. Here $\mathcal{C}^t = \{C_1^t, \ldots, C_n^t\}$, where n represents total number of clusters at time-stamp t. This map corresponds to the edges of the network. The weight $w(\mathrm{V}^t(j,k))$ of an edge $\mathrm{V}^t(j,k)$ between two clusters C_j^t and C_k^{t+1} corresponds to the fraction of nodes in cluster C_j at time-stamp t and cluster C_k at time-stamp $t+1$ which are assigned to cluster C_k at time-stamp $t+1$ and is represented as:

$$w(\mathrm{V}^t(j,k)) = \frac{|C_j^t \cap C_k^{t+1}|}{|C_j^t \cup C_k^{t+1}|} \tag{1}$$

where the numerator is equal to the number of nodes of cluster C_j^t which are also part of C_k^{t+1} and $|\ C_j^t \cup C_k^{t+1}\ |$ represents the total number of distinct nodes in clusters C_j^t and C_k^{t+1}. This fraction is equivalent to the widely-adopted Jaccard co-efficient [31]. This edge-weighting scheme is similar to the one proposed in [22] and was shown to successfully capture significant events. This edge weight calculation scheme is another method to track evolution of communities between 2 time-stamps and is different from the one proposed in [29]. It gives importance to both the nodes in C_j^t at time-stamp t and the nodes in C_k^{t+1} at time-stamp $t+1$.

An edge exists between two clusters C_j^t and C_k^{t+1} only if $w(\mathrm{V}^t(j,k)) > 0$. We then create an empty list L^t. Here we keep the information about the maximum weighted outgoing edge from each cluster C_i^t at time-stamp t i.e. $\mathrm{argmax}_j\, w(\mathrm{V}^t(i,j))$ and the maximum weighted incoming edge for each cluster C_j^{t+1} at time-stamp $t+1$ i.e. $\mathrm{argmax}_i\, w(\mathrm{V}^t(i,j))$. The list L becomes:

$$L^t = [w(\mathrm{V}^t(i, \mathrm{argmax}_j w(\mathrm{V}^t(i,j)))), \ldots, w(\mathrm{V}^t(\mathrm{argmax}_i w(\mathrm{V}^t(i,j)), j)), \ldots],$$

where $i = 1, \ldots, |\mathcal{C}^t|$ and $j = 1, \ldots, |\mathcal{C}^{t+1}|$. Here i and j vary from 1 to the total number of clusters

in the set $\mathcal{C}^t$ and $\mathcal{C}^{t+1}$ respectively. List L^t has all the maximum weighted edges from all $C_i^t \subset \mathcal{C}^t$ and all the maximum weighted edges to $C_j \subset \mathcal{C}^{t+1}$. We then define a threshold $\theta^t = L_k^t$ where $k = \text{argmin}_l L_l^t$, $l = 1, \ldots, |L|$ i.e. it is the minimum weight in the list L^t of maximum weighted edges in the bipartite graph W^t. We use this minimum weight θ^t to define the concept of majority or only those edges are kept in W^t whose edge weights are greater than or equal to θ^t. By selecting the minimum of all the maximally weighted edges (i.e. in-coming as well as out-going edges) between all the communities for 2 consecutive time-stamps, we even allow communities of smaller densities to have in-coming or out-going edges. If instead we take the average of all the maximally weighted edges for all the communities in these 2 time-stamps, we run into the risk of missing edges to and from communities of smaller density to communities of larger density. The average edge-weight criterion is influenced more by outliers or extreme edge-weights values. Thus, this criterion based on selecting the minimum of all the maximally weighted edges, acts an efficient tool a) prevent random noisy fluctuations from being detected as split or merge events and b) to detect various significant events for communities of different densities.

If the number of edges going out of a node of W^t is greater than 1, it indicates the possibility of a split at time $t + 1$, whereas if the number of edges entering a node of W^t at time $t + 1$ is greater than 1 then it indicates the possibility of a merge. A split will only happen if the corresponding clusters say C_k^{t+1} and C_l^{t+1} have an edge weight $w(\mathrm{V}^t(j, k)) \geq \theta^t$ and $w(\mathrm{V}^t(j, l)) \geq \theta^t$ respectively coming from cluster C_j^t. Similarly, a merge will only happen if weight of the edges from clusters say C_i^t and C_j^t to cluster C_k^{t+1} are greater than θ^t.

We observe continuation of a community when majority of the nodes from a cluster C_j^t are part of the some cluster at time $t + 1$ say cluster C_k^{t+1} i.e. $w(\mathrm{V}^t(j, k)) \geq \theta^t$. In order to tackle the death of one or more communities, we add a *Dump* cluster to the set of clusters $\mathcal{C}^t$ for each time-stamp t (except time 1 when we are first identifying the communities). The *Dump* cluster represents death of a community. Similarly, in order to handle birth of a new cluster from nodes which were previously unseen at time t, we add a cluster C_0^t to the set of clusters $\mathcal{C}^t$ for each time-stamp t except the first time-stamp. If more than one cluster is born at a given time-stamp t then we use identifiers C_0^t, C_{-1}^t, C_{-2}^t etc. for the newborn clusters. This allows to overcome the problem of detection of events like birth and death faced by the tracking algorithm in [29]. In [29], the authors used the same identifier for birth and death of communities which can lead to confusion when there is birth and death of a community at a given time-stamp. Moreover, this technique [29] cannot identify birth of multiple communities at a given time-stamp.

For the network W^t, if C_0^t is isolated then no new clusters were generated at time t and if C_0^t has an outgoing edge with weight $\geq \theta^t$ then a new community has emerged at time $t + 1$. Similarly, if we have incoming edges to *Dump* at time $t + 1$ in W^t then some clusters have disappeared. An advantage of having separate identifiers for tracking the birth and the death of communities is that it becomes easier to distinguish the two events when we are performing sql join operations [30] to generate the query database. Fig 3 gives an example of the mapping mechanism for 2 snapshots of Birthdeath dataset. This tracking procedure is summarized in Algorithm 1 as shown in Fig 4.

Visualizing Weighted Bipartite Networks

After obtaining the set $\mathcal{W}$, we visualize the weighted bipartite networks W^t for each time-stamp t. The steps involved in the visualization include using each bipartite network W^t as an adjacency matrix A^t and plotting the adjacency matrix as an image.

An edge in the adjacency matrix $A^t(i, j)$ indicates the connection between cluster C_i^t at time t and cluster C_j^{t+1} at time $t + 1$ and the weight of the edge is $A^t(i, j) = w(\mathrm{V}^t(i, j))$ obtained from

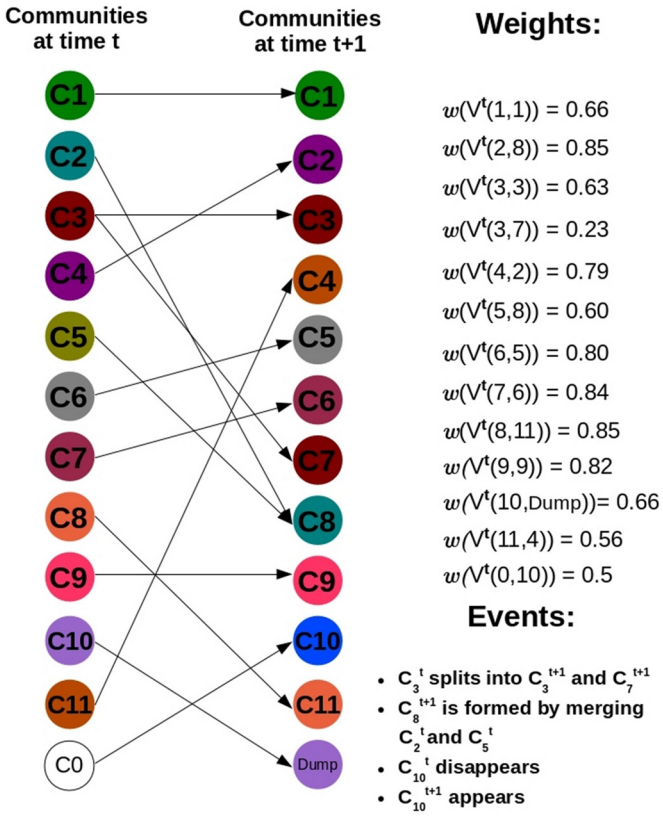


Fig 3. Weighted directed bipartite network W^t corresponding to 2 consecutive time-stamps for the synthetic Birthdeath dataset.

doi:10.1371/journal.pone.0137502.g003

W^t. For the purpose of visualization we remove all the edges directed to the *Dump* cluster. Let us say that cluster C_i^t disappeared at time $t + 1$. Then, sum of the weight of all the edges from community C_i^t to all the clusters at time-stamp $t + 1$ is 0. Hence, it becomes easier to identify communities which have disappeared. In the plot of A^t, we replace cluster C_i^t with *Dump*. In case of birth of one or more new communities we look up the entries corresponding to $A^t(0, i)$, $A^t(-1, j)$ etc. in the adjacency matrix at time-stamp $t + 1$. In the plot of A^t, we indicate these newborn clusters with the prefix 'NewC'.

An example of this visualization procedure is shown in Fig 5 in the case of a synthetic Hide dataset. This Hide dataset comprises 5 snapshots where one community disappears at each time-stamp t and one or more communities appear at each time-stamp $t > 1$. We use the evolutionary community detection algorithm introduced in [23] (namely Evolutionary Spectral Clustering) to illustrate that the proposed visualization also works for another evolutionary clustering algorithm.

Netgram Tool

Once we have obtained the set $\mathcal{W}$, we store the connections between nodes of the bipartite network W^t at time t in tabular format. For the sake of simplicity, we just keep the information about the source and the sink of the edge i.e. for an edge $V^t(j, k)$, we keep (j, k) in table P^t for time t. This results in a set of tables $\mathcal{P} = \{P^1, \ldots, P^{T-1}\}$ for T time-stamps.

Data: The set of clusters $\mathcal{C}^t$ and $\mathcal{C}^{t+1}$ at a given time-stamp t and $t+1$ respectively.
Result: A weighted directed bipartite network W^t tracking the relationship between the clusters at time-stamp t and $t+1$ and the minimum weight θ^t that an edge should satisfy in W^t.

$L^t = []$
foreach $C_j^t \in \mathcal{C}^t$ **do**
  foreach $C_k^{t+1} \in \mathcal{C}^{t+1}$ **do**
    if *Some Nodes with labels* C_j^t *at* t *have the label* C_k^{t+1} *at* $t+1$ **then**
      Create an edge $V^t(j,k)$ between C_j^t and C_k^{t+1}.
      $|C_j^t \cap C_k^{t+1}|$ = Number of nodes with labels C_j^t at t that have the label C_k^{t+1} at $t+1$.
      Weight of the edge $w(V^t(j,k)) = \frac{|C_j^t \cap C_k^{t+1}|}{|C_j^t \cup C_k^{t+1}|}$.
      Add the weighted edge $V^t(j,k)$ to the graph W^t.
    end
  end
  $k = \text{argmax}_k w(V^t(j,k))$.
  Append $w(V^t(j,k))$ to L^t. /* Add the edge weight to the list L^t. */
end
foreach $C_k^{t+1} \in \mathcal{C}^{t+1}$ **do**
  foreach $C_j^t \in \mathcal{C}^t$ **do**
    if *Some Nodes with labels* C_k^{t+1} *at time* $t+1$ *have the label* C_j^t *at time* t **then**
      Create and edge $V^t(j,k)$ between C_j^t and C_k^{t+1}.
      $|C_j^t \cap C_k^{t+1}|$ = Number of nodes with labels C_j^t at t that have the label C_k^{t+1} at $t+1$.
      Weight of the edge $w(V^t(j,k)) = \frac{|C_j^t \cap C_k^{t+1}|}{|C_j^t \cup C_k^{t+1}|}$.
      Add the weighted edge $V^t(j,k)$ to the graph W^t.
    end
  end
  $j = \text{argmax}_j w(V^t(j,k))$.
  if $w(V^t(j,k))$ *not in* L^t **then**
    Append $w(V^t(j,k))$ to L^t. /* Add the edge weight to the list L^t. */
  end
end
$\theta^t = L_k^t$ where $k = \text{argmin}_l L_l^t$, $l = 1, \ldots, |L|$.
/* Find the minimum weight of all the weights added to the list L^t. */
foreach *edge* $V^t(j,k) \in W^t$ **do**
  if $w(V^t(j,k)) < \theta^t$ **then**
    Remove edge $V^t(j,k)$ from W^t.
    /* Only keep significant edges in the weighted bipartite network W^t. */
  end
end

Fig 4. Algorithm 1: Community Tracking Algorithm.

doi:10.1371/journal.pone.0137502.g004

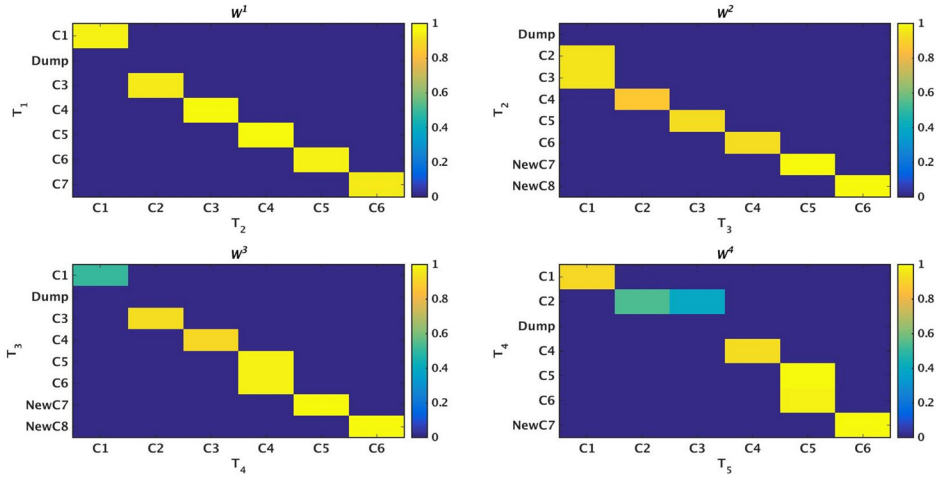


Fig 5. Visualization of the weighted networks W^t mapping the evolution of communities over 5 time-stamps. W^t tracks the evolution of a cluster between two successive time-stamps. We can observe that at each time-stamp T_i there is death of one community. We can also detect the birth of one or more communities at each time-stamp T_i, $i > 2$. The *x*-axis and *y*-axis represent the set of clusters at two consecutive time-stamps. The colors represent the weight of the edges for W^t.

doi:10.1371/journal.pone.0137502.g005

Table 1. Tracking the 7 communities detected by the OSLOM method [11] at time-stamp T1 for Mergesplit dataset. This tracking information is stored in database $\mathcal{D}$ which is depicted here.

T1	T2	T3	T4	T5
C_1	C_7	C_3	C_1	C_3
C_2	C_1	C_5	C_7	C_5
C_2	C_2	C_5	C_7	C_5
C_3	C_4	C_2	C_2	C_1
C_3	C_4	C_2	C_3	C_1
C_3	C_3	C_8	C_6	C_2
C_3	C_3	C_1	C_9	C_7
C_3	C_3	C_1	C_9	C_8
C_4	C_6	C_9	C_4	C_6
C_4	C_6	C_9	C_{10}	C_4
C_4	C_6	C_9	C_{10}	C_{11}
C_5	C_8	C_6	C_1	C_3
C_6	C_8	C_6	C_1	C_3
C_7	C_5	C_4	C_5	C_9
C_7	C_5	C_7	C_8	C_{10}

doi:10.1371/journal.pone.0137502.t001

We then construct a query database $\mathcal{D}$ by performing an outer-join sql operation [30, 32] between table P^1 and P^2 using the unique identifiers in 2^{nd} column of P^1 and 1^{st} column of P^2 as keys. We then repeat the process with query database $\mathcal{D}$ and succeeding tables in set $\mathcal{P}$ i.e. $P^3, \ldots, P^{T-1}$ using the unique identifiers of last column of database $\mathcal{D}$ and 1^{st} column of table P^i ($i > 2$) as keys on which the outer-join operation [30] is performed. By keeping separate unique identifiers for a dead cluster and a newborn community, it becomes easier to track birth and death of communities over time.

An example of the query database $\mathcal{D}$ obtained as a result of this process is shown in Table 1 for a synthetic Mergesplit dataset using the OSLOM [11] method. Since the OSLOM method is a hierarchical community detection algorithm, we only track large size communities at coarser levels of hierarchy for the 5 snapshots of the Mergesplit dataset.

We use a simple line-based tracking mechanism to visualize the query database $\mathcal{D}$. This line-based tracking is inspired by dendograms [33] which are used for visualization of layers of hierarchy in hierarchical clustering algorithms [15, 34, 35]. Similarly, the concept can be applied in case of dynamic networks where the layers represent the individual time-stamps for the evolving network.

While building the Netgram tool, we considered the following design principles:

- Each community is represented by a circle whose size is proportional to the number of nodes in that community at a given time-stamp t.
- The evolution of communities between 2 time-stamps is represented by a dashed line.
- Lines follow a straight path if there is no merge or split event during the entire course of evolution for a given community.
- If a part of a community say C_i^t merges into another community say C_j^{t+1} i.e. a merge event has occurred at time $t + 1$ then a dashed line is drawn from C_i^t at time t to the community it merges at time $t + 1$.

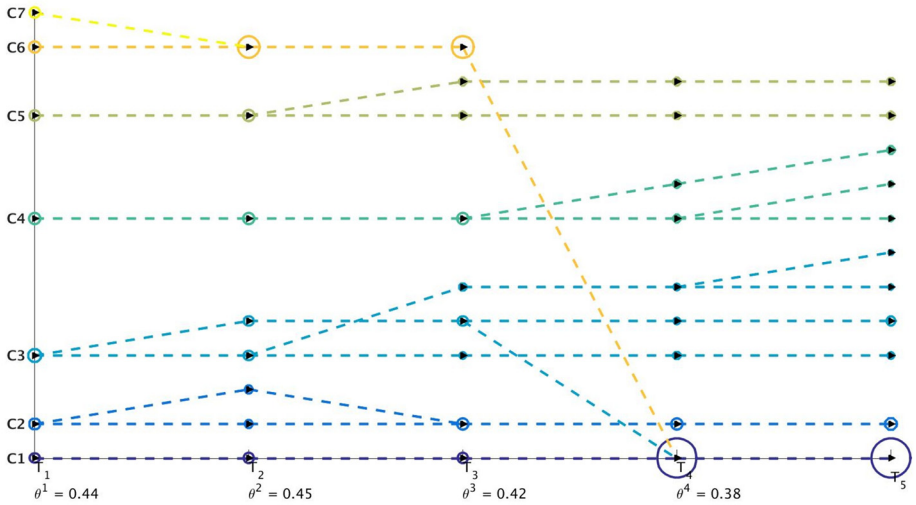


Fig 6. Visualization of evolution of communities obtained from OSLOM [11] method for Mergesplit dataset using Netgram. There are 11 cross-overs in this figure which is generated by passing the original order of partitions i.e $\mathcal{O} = \{1, 2, 3, 4, 5, 6, 7\}$ to Algorithm 2.

doi:10.1371/journal.pone.0137502.g006

- If a part of a community say C_i^t splits into another community say C_j^{t+1} i.e. a split event has occurred at time t then a dashed line is drawn from C_i^t at time t to the community it merges at time $t + 1$.

We also take into account the following aesthetic conditions as suggested in [36] when designing a visualization tool:

- Minimize line cross-overs.
- Minimize screen space.

These layout guidelines lead to a combinatorial problem with respect to the aesthetic conditions under the constraints of the design principles [36]. A similar problem for storyline visualization for streaming data was proposed in a constrained optimization framework in [36, 37]. In [36, 37] the authors try to visualize individuals (or groups) in a storyline. However, there is no event like merging or splitting of groups rather the groups come close together or move far from each other. Moreover, we can display additional information like size/density of communities at different time-stamps from the size of the circle representing those communities at these time-stamps. Fig 6 visualizes the evolution of communities over 5 time-stamps for the synthetic Mergesplit dataset using the community information obtained from OSLOM [11] method along with the original order of clusters in $\mathcal{C}^1$ by providing this information to Algorithm 2 (Fig 7).

The evolutionary clustering algorithm initially provides the community labels for all the nodes at a given time-stamp t. Using the query database $\mathcal{D}$ and the initial order of the partitions in $\mathcal{C}^1$ i.e. $\mathcal{O} = \{1, 2, \ldots, n\}$ where n represents the total number of communities in the 1^{st} time-stamp, we propose a simple algorithm explained in Fig 7 that satisfies all the design principles and the aesthetic criteria except the minimization of line cross-overs. By using line based visualization instead of surface based visualization we minimize the screen space.

We now explain the concept of a line cross-over. A line cross-over generally occurs in case of a split or merge event. For example, in case of Fig 6, community **C6** merges into community

Data: Query Database $\mathcal{D}$, total number of time-stamps T and the order of the clusters i.e. $\mathcal{O}$.
Result: Line-based visualization satisfying the design principles.

$i = 1, y = 0$, $n = |\mathcal{O}|$, $M = \{M_2, \ldots, M_T\}$
/* M is a memory-map which keeps information about the communities present at all the time-stamps t from $t = 2, \ldots,$ T. Initialize each of M_t as an empty list. */
while $i \leq n$ **do**
  $cid = \mathcal{O}_i$. /* Select the i^{th} id from $\mathcal{O}$ to represent which community will be visualized at the i^{th} iteration. */
  $j = 2$ & draw a circle at co-ordinates $(j-1, y)$ representing community C_{cid} $\propto$ its size.
  $List_{cid} = \{C_{cid}\}$. /* Add C_{cid} to a list. */
  while $j \leq T$ **do**
    while $List_{cid} \neq []$ **do**
      C_{cid} = Pop first element from $List_{cid}$.
      R = Result of the query to $\mathcal{D}$ finding the rows corresponding to C_{cid} at time $j-1$.
      $SubC_{cid}$ = Unique list of communities connected to C_{cid} at time j obtained from R.
      foreach $c_k \in SubC_{cid}$ & $c_k \notin M_j$ **do**
        if $c_k = Dump$ **then**
          Stop the time-line for C_{cid}. /* Handles the death event. */
        end
        else
          Draw a circle at (j, y) representing community c_k $\propto$ its size.
          Add c_k to M_j.
          Draw a dashed line from C_{cid} at time $j-1$ to c_k at time j.
          $y := y + 1$.
        end
      end
      /* Handles the growth, shrinkage, continuation and split events. */
      foreach $c_k \in SubC_{cid}$ & $c_k \in M_j$ **do**
        Draw a dashed line from C_{cid} at time $j-1$ to an existing community c_k at time j.
      end
      /* Handles the merge event. */
    end
    $j := j + 1$ & $List_{cid} = SubC_{cid}$.
  end
  $List_{cid} = \{\}$ & $i := i + 1$ & $y := y + 1$.
end
For new communities follow the same procedure as above using the memory-map M generated from the existing communities and query database $\mathcal{D}$. /* Handles birth event. */

Fig 7. Algorithm 2: Netgram Visualization Layout.

doi:10.1371/journal.pone.0137502.g007

C1 at time-stamp T_4. However, the line showing the merge event crosses over the time-line of communities **C5, C4, C3, C2** and its branches i.e. 8 lines in total. Similarly, community **C3** has one cross-over at time-stamp T_3 during a split event and 2 cross-overs at time-stamp T_4 due to a merge event with community **C1**. Thus, in total there are 11 cross-overs using the initial order of clusters i.e. $\mathcal{O}$ = {1, 2, 3, 4, 5, 6, 7}. However, if we place **C6** next to cluster **C1** and community **C7** next to **C6** and maintain the order of the remaining communities i.e. $\mathcal{O}$ = {1, 6, 7, 2, 3, 4, 5}, we can already reduce the number of line cross-overs to 3. We provide a greedy solution in Algorithm 3 (Fig 8) to sequence the order of clusters/partitions in $\mathcal{C}^1$ such that there are fewer line cross-overs in comparison to the initial order of the clusters i.e. $\mathcal{O}$ = {1, . . ., n} provided by the evolutionary community detection technique.

Fig 9 illustrates the effect of applying Algorithm 3 on the initial order of partitions in $\mathcal{C}^1$ obtained from the OSLOM method [11] for the Mergesplit dataset and providing this new order $\mathcal{O}$ to Algorithm 2 to perform visualization using Netgram. Fig 10 summarizes the steps undertaken by the Netgram tool for visualization of evolution of communities.

Additional Provisions

We provide the user with some additional facilities. The user can provide two parameters ρ and ν as input. The former specifies the minimum weight for an edge in the bipartite network W^i for all time-stamps i = 1, . . ., T-1. Netgram will then visualize only those edges whose weights are greater than or equal to ρ. This allows the user to interact with the Netgram tool

Data: Original order of clusters $\mathcal{O}$, total number of time-stamps T and query database $\mathcal{D}$.
Result: New order of partitions $\mathcal{O}$

$i = 2$, $\mathcal{S} = \{\}$, $M = \{\}$ and $Queue = L = []$. /* Initialize an empty map M which will be used to map for each cluster $c_j \in \mathcal{C}^1$ the sequence of clusters which had interaction with c_j at different time-stamps. */
while $i \leq T$ **do**
  R = Result of the query to $\mathcal{D}$ to find which community or communities $c_j \in \mathcal{C}^1$ at time $i-1$ has experienced a merge or a split event from which community or communities $c_k \in \mathcal{C}^1$ at time i.
  Generate tuples (c_j, c_k, i) from R and append to $Queue$.
end
while $Queue \neq []$ **do**
  P = Pop first tuple out of queue. /* This allows us to handle communities which merged or splitted at earlier time-stamps first. */
  Obtain c_j and c_k and corresponding ids j, k from tuple P.
  if $k \notin M(j)$ **then**
    Append k to the sequence of j.
  end
end
/* M maps for each cluster $c_j \in \mathcal{C}^1$ the clusters $c_k \in \mathcal{C}^1$ with which it interacted ordering them based on the time-stamp of interaction i.e. the cluster with which c_j interacted first is placed first in the $M(j)$ and the cluster with which it interacted last is placed last. */
foreach $j \in \mathcal{O}$ **do**
  Insert j and all elements of $M(j)$ to L (j appearing first). /* $j = 1, \ldots, n$ */
end
foreach $j \in L$ **do**
  if $j \notin \mathcal{S}$ **then**
    Append j to $\mathcal{S}$.
  end
end
/* $\mathcal{S}$ is the greedy solution to handle cross-overs. It uses the simple principle to keep together those clusters which had interacted earlier in time in order to reduce cross-overs. */
$\mathcal{O} = \mathcal{S}$.

Fig 8. Algorithm 3: Greedy Solution to Handle Cross-Overs.

doi:10.1371/journal.pone.0137502.g008

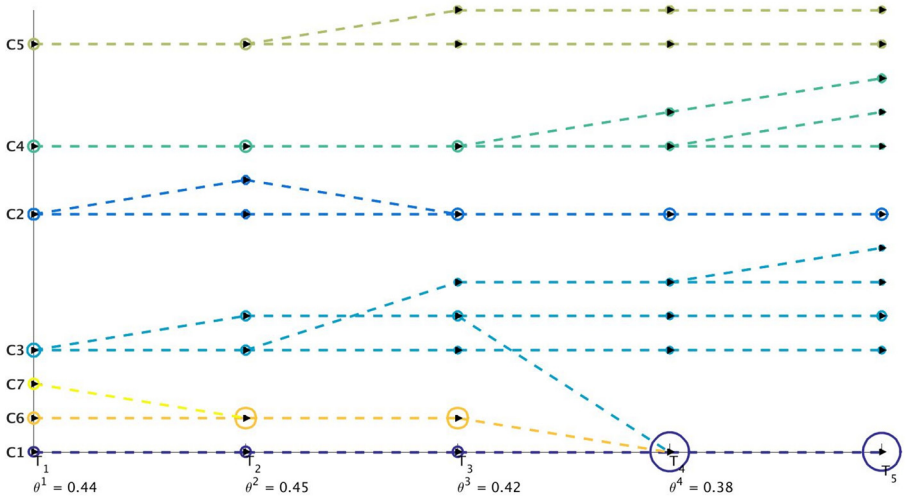


Fig 9. Visualization of evolution of communities for Mergesplit dataset using the refined order of partition $\mathcal{O}$ obtained from Algorithm 3 and then passed to Algorithm 2. The revised ordering of partitions results in just 2 cross-overs as depicted in this figure.

doi:10.1371/journal.pone.0137502.g009

Data: Node identifiers and community affiliation for nodes appearing in a dynamic network. The community affiliation can be obtained from any evolutionary clustering algorithm.
Result: A visualization of the evolution of communities over time.
1 Apply the modified version of Tracking Algorithm 1.
2 Generate a set $\mathcal{W}$ comprising weighted directed bipartite networks W^t capturing the evolution of communities between 2 successive time-stamps.
3 Visualize these weighted bipartite networks (W^t).
4 Store these weighted bipartite networks (W^t) into tables P^t.
5 Perform sql outer-join operation [30, 31] on these tables to get a query database $\mathcal{D}$.
6 Re-order the initial partition $\mathcal{C}^1$ provided by evolutionary community detection algorithm to have few cross-overs using Algorithm 3.
7 Visualize $\mathcal{D}$ satisfying all design principles and greedily fulfilling aesthetic guidelines using Algorithm 2.

Fig 10. Steps undertaken by the Netgram tool for visualization.

doi:10.1371/journal.pone.0137502.g010

and focus on certain communities which are less prone to random noisy events and have edges with weights above ρ during their entire evolution period.

The latter (ν) is a threshold which defines the tolerance level allowed for ρ to differ from average θ^i for all the time-stamps $i = 1, \ldots, \text{T-1}$. Average θ^i is defined as $\mu_\theta = \frac{\sum_{i=1}^{\text{T}-1} \theta^i}{\text{T}-1}$. If $|\rho - \mu_\theta| \leq \nu$, we set $\theta^i = \rho$ for all time-stamps else we maintain the original value of θ^t as obtained from Algorithm 1. This tolerance level prevents removal of too many edges from each W^i and prevents generating a near empty visualization plot. In our experiments we set the tolerance level value $\nu \leq 0.1$ thus not allowing too much variation between user-specified ρ and μ_θ but at the same time allowing some non-essential edges to be removed from the visualization tool.

We also provide the user with an additional facility which allows to visualize the network configuration at a given time-stamp t. For each time-stamp t we plot the network using the community information and the edge flow between the communities. Each community is plotted as a circular disc and the size of this circular disc is proportional to the number of nodes within the community. These communities are connected to each other using edges. The weight of the edges is proportional to the total number of edges flowing from one community to another. These edge weights are normalized to take a values between [0, 1]. This is done by taking the ratio of the number of edges flowing between 2 communities to the maximum number of edges flowing between any 2 communities among the set of communities at this given time-stamp t. The edges are displayed as lines connecting 2 communities and are plotted in gray-scale format. Edges with weight close to 0 are represented with whiter shades whereas those closer to 1 as drawn with darker shades of gray. In each row of the plot we can showcase the network configuration for a maximum of 5 time intervals. Using this visualization technique, we can observe significant events like birth, death, growth, shrinkage, continuation and split. However, the visualization of a merge event is not feasible in this scheme.

We illustrate the usage of these parameters ρ and ν for 3 settings in case of Mergesplit dataset using the community information obtained from Louvain method. We keep the ν value fixed at 0.1 and set $\rho \in \{0.4, 0.45, 0.5\}$ to obtain results as depicted in Figs 11, 12, 13, 14, 15 and 16 respectively.

Experiments

We first provide a brief description of the datasets used in this paper. In the previous sections, we have seen the use of several synthetic datasets including:

- **Birthdeath**—A dataset comprising 5 dynamic networks. There are 13 communities in the network at time-stamp t_1. At every time-stamp t_i, $i > 2$ there is death of one community and

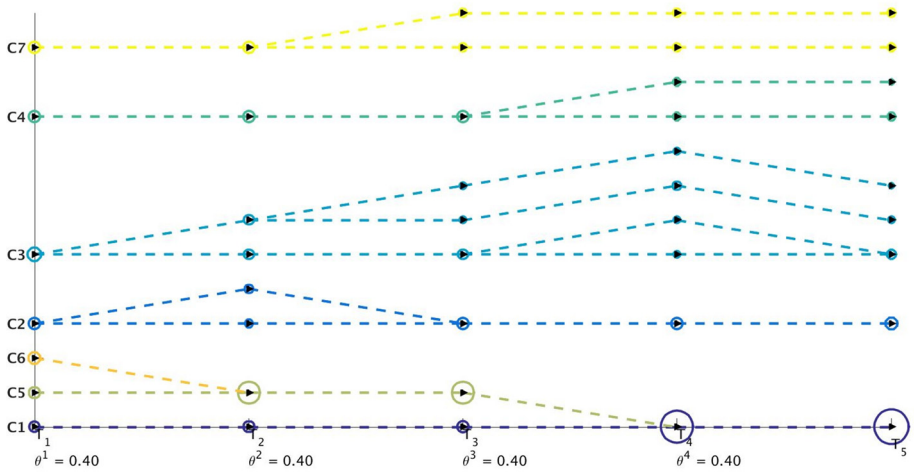


Fig 11. Visualization of communities for Mergesplit dataset obtained by Louvain [5] method keeping $\rho = 0.4$ and $v = 0.1$. The value of $\mu_\theta = 0.42$ and since $|\rho - \mu_\theta| < v$, so we use ρ as the minimum weight of an edge for all time-stamps $t = 1, \ldots, T\text{-}1$ as depicted in the figure. We observe that most of the edges are retained for this value of ρ as all edge weights are $w(V^t(i, j)) \geq \rho$.

doi:10.1371/journal.pone.0137502.g011

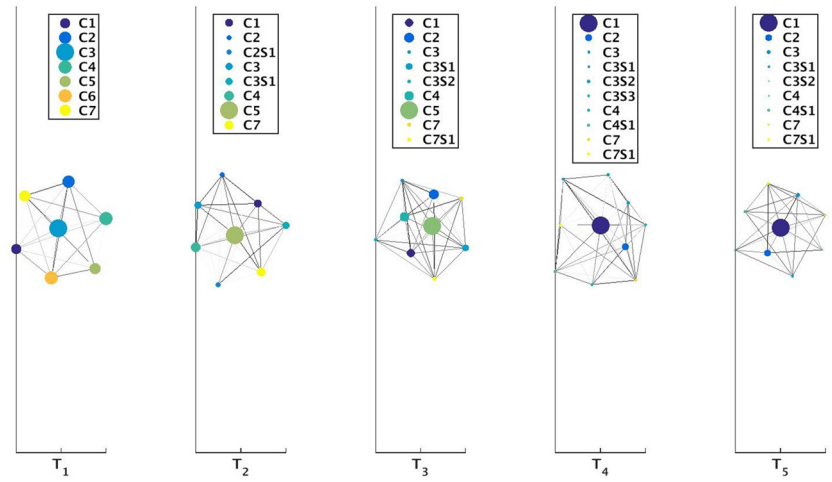


Fig 12. Visualization of network for Mergesplit dataset corresponding to Fig 11. The circular discs represent the communities with size proportional to number of nodes in these communities. From the legend of the subplot at time-stamp T_2 we observe that community **C2** had experienced a split event at time-stamp T_1 generating communities **C2** and **C2S1**. Here **CiSj** represents $(j + 1)^{th}$ community appearing from **Ci**. Darker shaded edges represent communication between 2 clusters.

doi:10.1371/journal.pone.0137502.g012

at each time-stamp $t_i > 3$ there is birth of one community. The number of nodes in the networks decrease from 1,000 to 886 over time. We illustrate the evolution of communities for this dataset using the MKSC algorithm [28, 29] in Fig 2.

- **Hide**—A dataset comprised of 5 dynamic networks. There are 7 communities in the network at time t_1. At each time-stamp t_i, one community disappears as was shown in the visualization of the weighted bipartite networks in Fig 5.

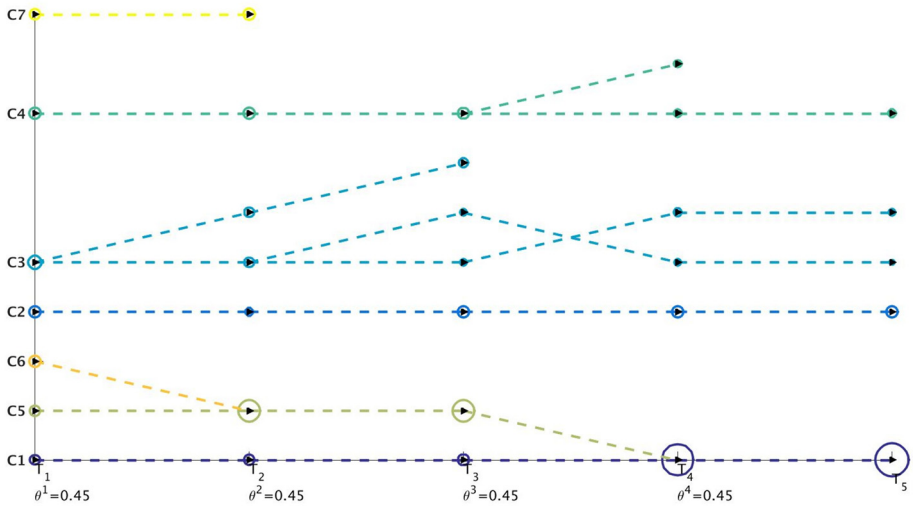


Fig 13. Visualization of communities for Mergesplit dataset obtained by Louvain [5] method keeping ρ = 0.45 and v = 0.1. We observe that several edges have been removed in comparison to Fig 11. The life time of community **C7** has been shortened to just 2 time-stamps and some edges with weights less than ρ have been removed from community **C3**.

doi:10.1371/journal.pone.0137502.g013

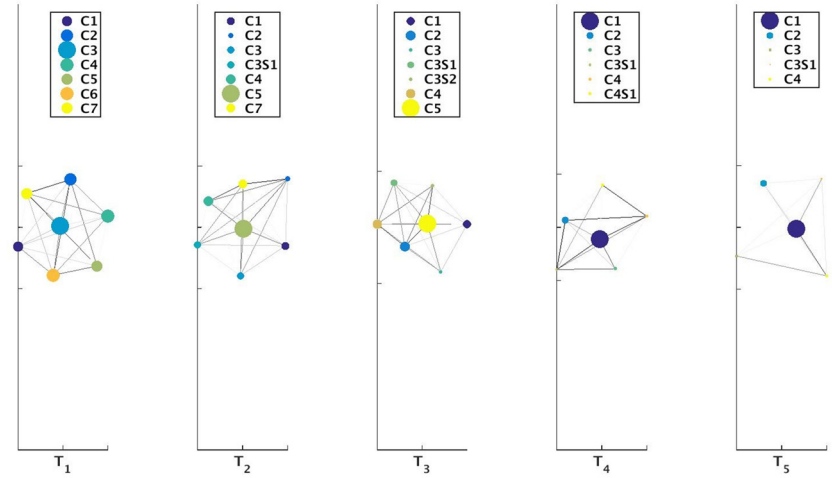


Fig 14. Visualization of network for Mergesplit dataset corresponding to Fig 13. The circular discs represent the communities with size proportional to number of nodes in these communities.

doi:10.1371/journal.pone.0137502.g014

- **Mergesplit**—A dataset comprising 5 dynamic networks. There are 7 communities in the network at time t_1 and 1,000 nodes in each of the 5 snapshots of this dataset. At each time-stamp t_i, there is 1 merge and 2 split events. A visualization of the evolution of the communities obtained by OSLOM [11] method for this dataset was provided in Fig 6.

These networks were generated using the software available at https://github.com/derekgreene/dynamic-community. We also experimented on real-life datasets and a synthetic large scale dataset (to show the scalability of the Netgram tool) which are described below:

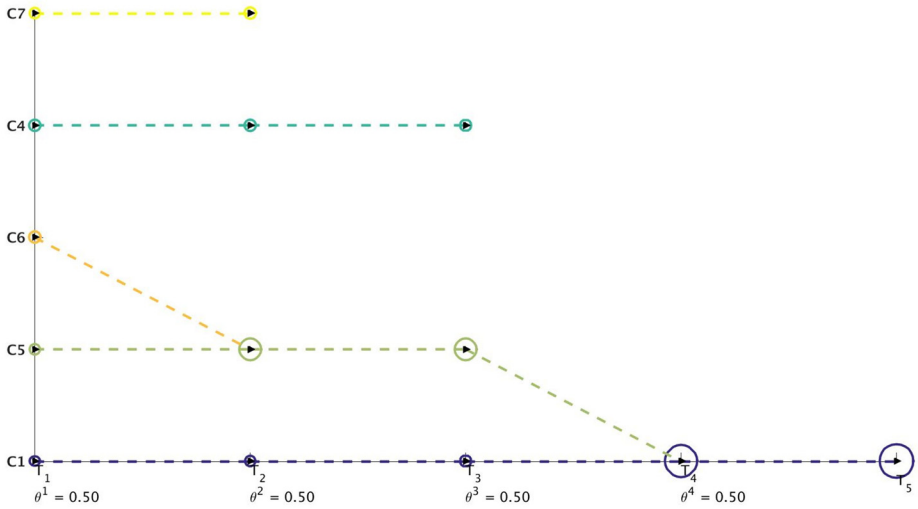


Fig 15. Visualization of communities for Mergesplit dataset obtained by Louvain [5] method keeping ρ = 0.5 and v = 0.1. We observe that when $\rho = 0.5$ most of the edges are removed for Mergesplit dataset. Communities **C2** and **C3** no longer are part of the palette and life time of communities **C4** and **C7** are shortened. We can also observe that communities **C1**, **C5** and **C6** remain unchanged in comparison to Figs 11 and 13 indicating these communities have more stable evolution.

doi:10.1371/journal.pone.0137502.g015

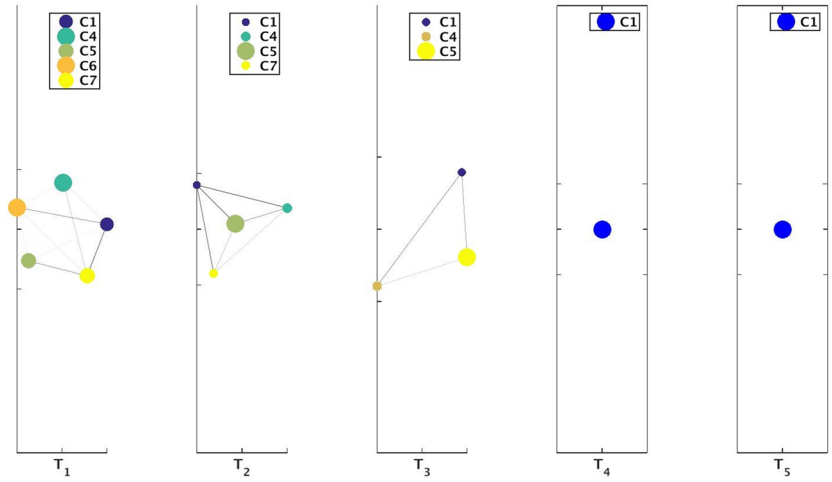


Fig 16. Visualization of network for Mergesplit dataset corresponding to Fig 15. The circular discs represent the communities with size proportional to number of nodes in these communities. There exists only community **C1** for time-stamp T_4 and T_5 using $\rho = 0.5$. If only one community is present then it is visualized using a blue-colored disc.

doi:10.1371/journal.pone.0137502.g016

- **Reality**—This dataset monitors the cellphone activity of people from 2 different labs in MIT [38]. This dataset consists of networks where the node represents a user whose cellphone periodically scans for other cellphones via Bluetooth. The weight of the edge between two nodes is equal to the number of intervals for which the 2 users are in close proximity to each other. Each snapshot corresponds to a weighted network which records the activities of the users over a period of 1 week. There is a total 32 meaningful snapshots and total number of users monitored over this time-span is 94. However, not all users are present in each

weighted network. The smallest network comprised of 21 people and largest network had 88 people.

- **NIPS**—This dataset consists of information about 1,500 papers published in the Advances in Neural Information Processing Systems (NIPS) conference starting from 1987 to 1998. The dataset is part of the Bag of Words Dataset [39] from the UCI repository http://archive.ics.uci.edu/ml/. From this dataset, we first separate out the papers published in each year starting from 1987 till 1998. We then create a TF-IDF model [40] using which we remove out irrelevant words from the documents. We then create a word-word graph for each time-stamp where the weight of the edges in the network is proportional to occurrence of 2 words together in all the documents for that year.
- **Weather**—The website http://www.wunderground.com/ was utilized to obtain weather information for about 9 months for 23 European cities. The time period over which this data spanned was from January 2012 to October 2012. For each city for each day we collected information about 20 attributes. The goal was to cluster these cities using weekly information about these cities i.e. one snapshot consists of 23 cities and 140 variables where 20 variables from each day are concatenated together. Thus in total we have 40 snapshots corresponding to 40 weeks.
- **Big**—This dataset consists of 5 networks where the network at the first time-stamp has 1 million nodes. There are 10 communities in the network at time t_1. The number of nodes decrease over time. The dataset exhibits several significant events like merge, split and death of communities. It was also generated from the software provided in [22].

The Netgram toolkit is most suitable for visualization of a small number of large sized clusters. This is because in case of a large number of clusters the palette for plotting will become too cluttered and it becomes difficult to keep track of the evolution of individual communities. Thus, in case of a hierarchical evolutionary clustering algorithm like OSLOM [11] or Louvain [5] method, Netgram is most suitable for visualization of giant-connected components at coarser levels of hierarchy.

Figs 17 and 18 depict the evolution of communities for the Reality and the Big dataset respectively. In case of the Reality dataset, we use the Evolutionary *k*-Means technique introduced in [23] to obtain the community affiliation for all the nodes at different time-stamps. In case of the Big dataset, we use the Louvain method [5] to generate the community memberships for all the nodes in the network over different periods of time. We use the Louvain method as it can easily scale to 1 million nodes for community detection which is otherwise a difficult task for evolutionary clustering algorithms.

NIPS Results

We obtain the communities from the word-word graph for the NIPS dataset using the MKSC [28] algorithm. The MKSC technique identified 5 communities at each time-stamp for this dataset. However, as we observe from Fig 19, the birth of several new communities was detected by the MKSC algorithm. We observe the appearance of disciplines like Speech Recognition and Computer Vision as distinguishable and distinct communities (in comparison to Supervised Learning Techniques) in 1991 and 1992 respectively. Fig 19 illustrates that the Netgram tool also detected the inception of NIPS Workshops in the year 1993. By the year 1998, we have specialized disciplines like Neural Networks, Robotics, Kernel Methods and Bayesian Methods which are combinations of Supervised Learning Techniques and Unsupervised Learning Techniques as observed from Fig 19.

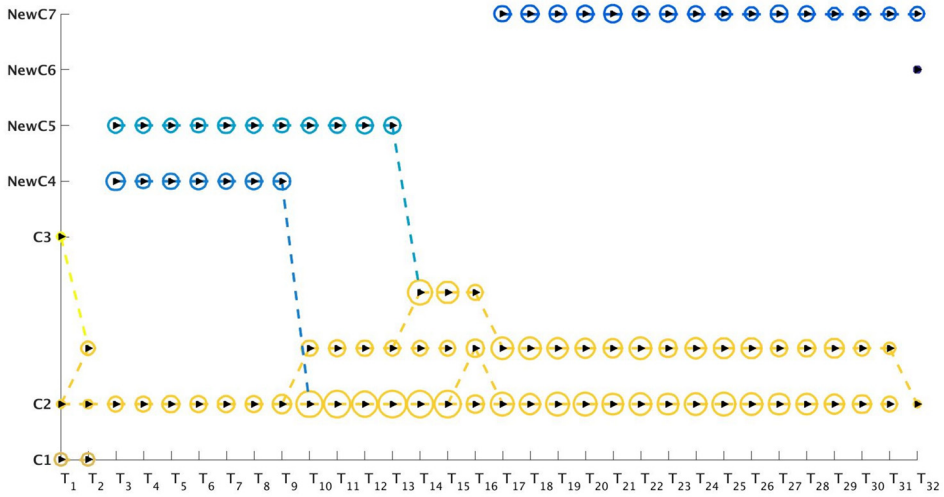


Fig 17. Visualization of evolution of communities generated by Evolutionary *k*-means [23] method using the Netgram tool for the Reality dataset. We mostly observe continuation of communities. However, communities **C1** and **C3** both disappear at time-stamp T_2 and new communities **NewC4** and **NewC5** appear at time-stamp T_3. By time-stamp T_{14} all communities merge into branches of community **C2**. New communities **NewC6** and **NewC7** appear at time-stamps T_{18} & T_{32} respectively.

doi:10.1371/journal.pone.0137502.g017

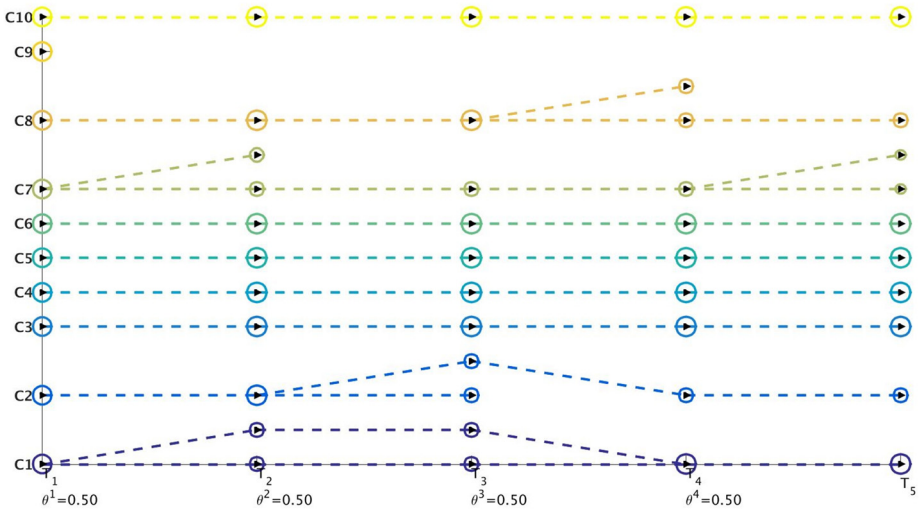


Fig 18. Visualization of evolution of communities generated by the Louvain [5] method using the Netgram tool for the Big dataset. We can observe significant events like merge, split, death and shrinkage of communities. For example, we can observe the split of communities **C7** and **C8** at time-stamp T_2 and T_3 respectively. Similarly, we can observe a merge event for cluster **C1** at time T_4. We observe a general pattern of shrinkage for most of the communities in this dataset.

doi:10.1371/journal.pone.0137502.g018

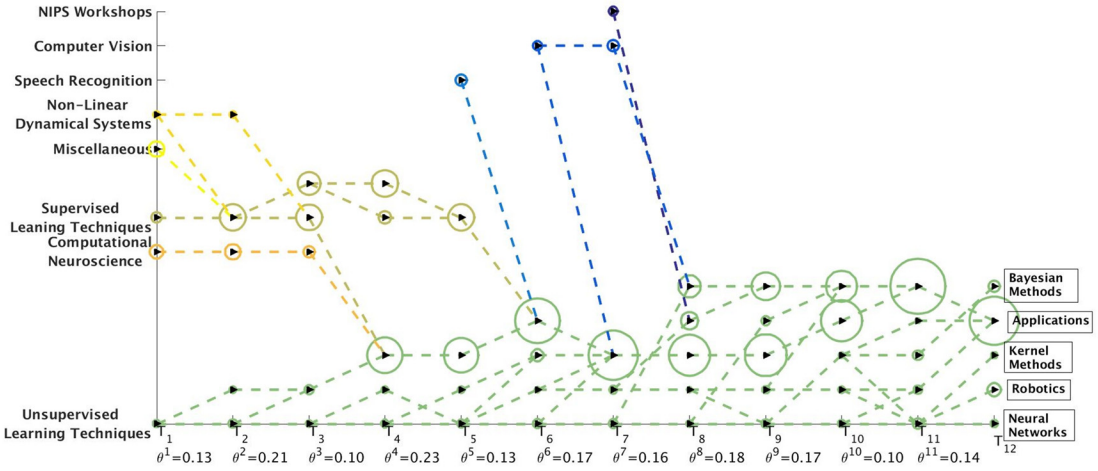


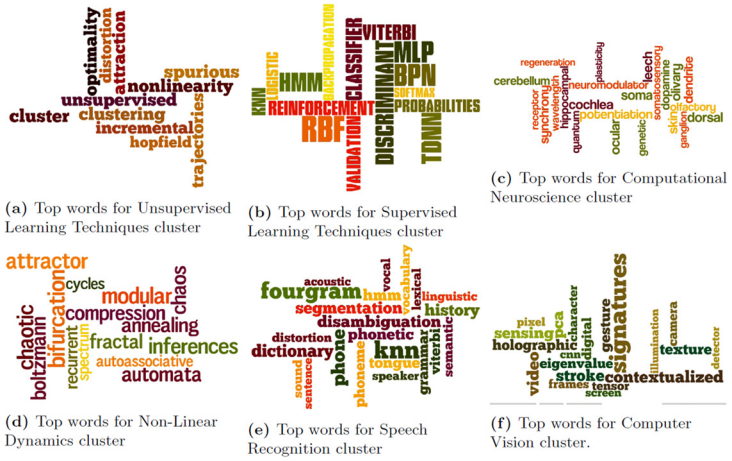
Fig 19. Visualization of evolution of communities for the NIPS dataset by Netgram toolkit.

doi:10.1371/journal.pone.0137502.g019

Since we use a TF-IDF model [40] to obtain the word-word graph on which we perform the community detection for the NIPS dataset, we can identify the top words (based on TF-IDF [40]) in these communities. In Figs 20 and 21, we showcase these top words corresponding to the communities detected by MKSC algorithm [28, 29].

Weather

For the weather dataset for each day we gathered information about attributes like maximum, minimum and mean temperature, dew point, humidity, sea level pressure, visibility, wind speed respectively and precipitation and wind direction for each city. The 23 European cities for which the data was gathered included Amsterdam, Antwerpen, Athens, Berlin, Brussels, Dortmund, Dublin, Eindhoven, Frankfurt, Groningen, Hamburg, Liege, Lisbon, London,



(a) Top words for Unsupervised Learning Techniques cluster

(b) Top words for Supervised Learning Techniques cluster

(c) Top words for Computational Neuroscience cluster

(d) Top words for Non-Linear Dynamics cluster

(e) Top words for Speech Recognition cluster

(f) Top words for Computer Vision cluster.

Fig 20. Word clouds for top words in the communities detected by MKSC method [28, 29] over the first few time-stamps for the NIPS dataset.

doi:10.1371/journal.pone.0137502.g020

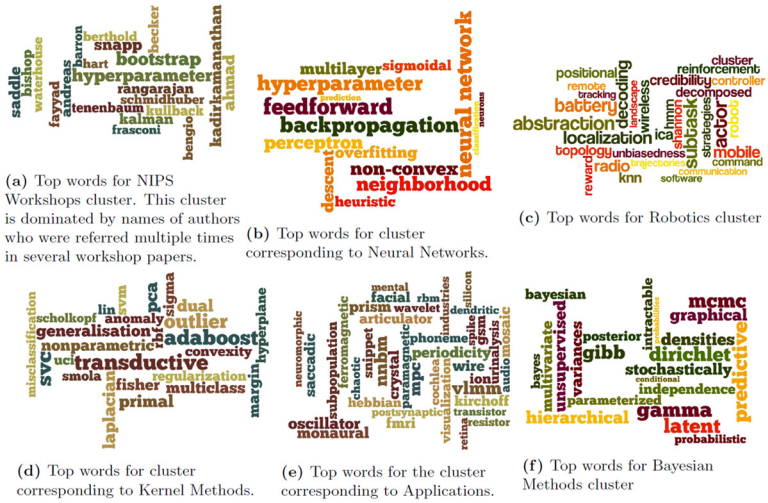


Fig 21. Word clouds for top words in the communities detected by MKSC method [28, 29] over the last few time-stamps for the NIPS dataset.

doi:10.1371/journal.pone.0137502.g021

Madrid, Milan, Nantes, Paris, Prague, Rome, Toulouse, Vienna and Zurich. We run the MKSC [28, 29] algorithm to obtain the communities for different time-stamps. The MKSC technique takes into account temporal smoothness [19] using the memory. Fig 22 depicts the result obtained for the weather dataset using MKSC algorithm.

Initially community **C1** consists of all the 23 European cities. However, after beginning of April (T_{13}) cluster **C1** splits into 2 or more prominent communities for all later time-stamps. This suggests that winter pattern in most of the European cities are nearly same. During the first week of April (T_{13}) the smaller cluster comprises cities like Milan, Rome, Nantes, Paris, Prague, Toulouse and Vienna which indicated that outbreak of spring in these cities are similar to each other and different from other European cities. During mid May (T_{19}), we observe 3 communities where one cluster was just the city of Dublin, the other 2 clusters consisted of Amsterdam, Antwerpen, Brussels, Berlin, Dortmund, Eindhoven, Frankfurt, Groningen, Hamburg, Liege, London, Nantes, Paris and Athens, Lisbon, Madrid, Milan, Prague, Rome, Toulouse, Vienna, Zurich respectively. This clustering information clearly distinguishes the weather pattern of western European cities from cities of southern Europe (except Prague).

The Netgram tool is available for usage at http://www.esat.kuleuven.be/stadius/ADB/mall/downloads/Netgram_Tool.zip.

Conclusion

In this paper we proposed a visualization toolkit Netgram which can be used to depict the evolution of communities in dynamic networks over time. Netgram was used to illustrate the occurrence of significant events like birth, death, merge, split, expansion, shrinkage and continuation of communities over time. Netgram can be used as a post-processing step to any evolutionary clustering algorithm. Netgram provides a simple line-based visualization tool for tracking the evolution of communities for various datasets. The tracking of the evolution of communities was performed in such a way that there were only a few line cross-overs and efficient screen space usage (since we used dashed lines). We proposed a greedy solution to have

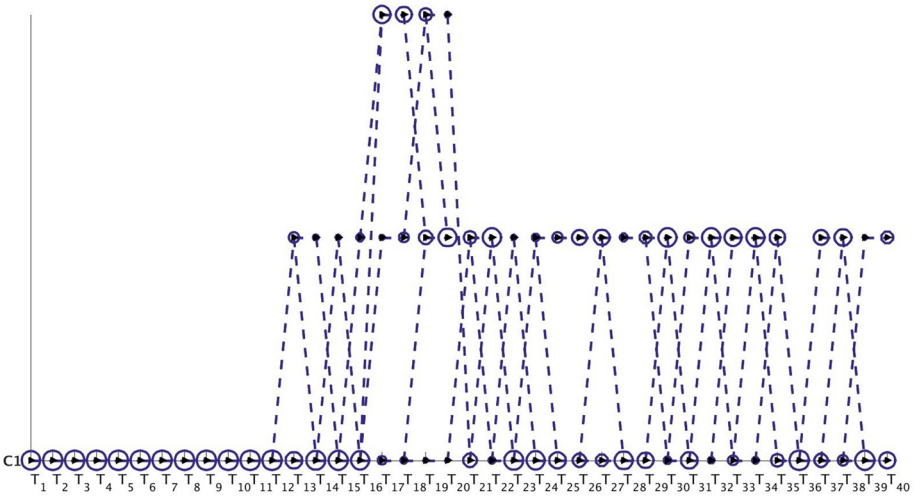


Fig 22. Visualization of evolution of clusters obtained using MKSC [28] algorithm for the weather dataset by Netgram toolkit. The y-axis showcases that all the 23 European cities belong to one cluster **C1** at time T_1. The cluster **C1** splits into 2 or more clusters at later time-stamps. But, since all these splitted clusters originate from **C1** and the set of cities (nodes) remain constant, they are plotted with the same colour as **C1**. We provided $\rho = 0.5$ and $v = 0.1$ as parameters to visualize only significant events and for all time-stamps we observe that $|\rho - \mu_\theta| \leq v$. Thus, $\theta^t = \rho = 0.5$ for all time-stamps $t < 40$ and is kept constant. So, we don't specify the constant threshold value on the x-axis and prevent the visualization palette from getting over-crowded.

doi:10.1371/journal.pone.0137502.g022

fewer line cross-overs in comparison to the plot obtained by using the original order or partitions ($\mathcal{O}$) as provided by the evolutionary clustering algorithm.

Acknowledgments

This work was supported by EU: ERC AdG A-DATADRIVE-B (290923), Research Council KUL: GOA/10/-/09 MaNet, CoE PFV/10/002 (OPTEC), BIL12/11T; PhD/Postdoc grants-Flemish Government; FWO: projects: G.0377.12 (Structured systems), G.088114N (Tensor based data similarity); PhD/Postdoc grants; IWT: projects: SBO POM (100031); PhD/Postdoc grants; iMinds Medical Information Technologies SBO 2014-Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO, Dynamical systems, control and optimization, 2012-2017).

Author Contributions

Conceived and designed the experiments: RM JS. Performed the experiments: RM. Analyzed the data: RM RL. Contributed reagents/materials/analysis tools: RM. Wrote the paper: RM JS.

References

1. Crandall D., Cosley D., Huttenlocher D., Kleinberg J. and Suri S. (2008) Feedback effects between similarity and social influence in online communities. In Proc. of KDD, pp. 160–168.
2. Jeong H., Tombor B., Albert R., Oltvai Z. and Barabási A. (2000) The large scale organization of metabolic networks, Nature 407 (6804):651–654. doi: 10.1038/35036627 PMID: 11034217
3. Barabási A., Jeong H., Neda E., Ravasz A. and Vicsek T. (2002) Evolution of social network of scientific collaborations. Physica A: Statistical Mechanics and its Applications, 311 (3-4):590–614. doi: 10.1016/S0378-4371(02)00736-7
4. Selcuk A. A., Uzun E. and Pariente M. R. (2008) A Reputation-based trust management system for p2p networks. I. J. Network Security, 6(2): 227–237.

5. Blondel V, Guillaume J, Lambiotte R, Lefebvre L. (2008) Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 10:P10008. doi: 10.1088/1742-5468/2008/10/P10008
6. Girvan M, Newman ME (2002) Community structure in social and biological networks. PNAS, 99 (12):7821–7826. doi: 10.1073/pnas.122653799 PMID: 12060727
7. Fortunato S. (2009) Community detection in graphs. Physics Reports 486:75–174. doi: 10.1016/j.physrep.2009.11.002
8. Clauset A, Newman ME, Moore C. (2004) Finding community structure in very large scale networks. Physical Review E, 70(066111).
9. Rosvall M, Bergstrom C. (2008) Maps of random walks on complex networks reveal community structure. PNAS, 105:1118–1123. doi: 10.1073/pnas.0706851105 PMID: 18216267
10. Lancichinetti A, Fortunato S. (2009) Community detection algorithms: a comparitive analysis. Physical Review E, 80(056117).
11. Lancichinetti A, Radicchi F, Ramasco J, Fortunato S. (2011) Finding statistically significant communities in networks. PLOS ONE, 6(e18961). doi: 10.1371/journal.pone.0018961 PMID: 21559480
12. Alzate C, Suykens JAK. (2009) Multiway spectral clustering with out-of-sample extensions through weighted kernel PCA. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(2):335–347. doi: 10.1109/TPAMI.2008.292
13. Mall R, Langone R, Suykens JAK. (2013) Kernel Spectral Clustering for Big Data Networks. Entropy (Special Issue: Big Data), 15(5):1567–1586.
14. Mall R, Langone R, Suykens JAK. (2013) Self-Tuned Kernel Spectral Clustering for Large Scale Networks. In Proceedings of the IEEE International Conference on Big Data (IEEE BigData 2013), pp. 385–393.
15. Mall R, Langone R, Suykens JAK. (2014) Multilevel Hierarchical Kernel Spectral Clustering for Real-Life Large Scale Complex Networks. PLoS ONE, 9(6): e99966. doi: 10.1371/journal.pone.0099966 PMID: 24949877
16. Mall R, Suykens JAK (2013) Sparse Reductions for Fixed-Size Least Squares Support Vector Machines on Large Scale Data. In Proceedings of 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013), pp. 161–173.
17. Xie J, Kelley S, Szymanski BK. (2013) Overlapping community detection in networks: the state of the art and comparative study. ACM Computing Surveys 45(4), Article 43. doi: 10.1145/2501654.2501657
18. Aynaud T, Fleury E, Guillaume JL, Wang Q. (2013) Communities in Evolving Networks: Definitions, Detection and Analysis Techniques. Dynamic on and of Complex Networks, Spring, vol 2, 159–200.
19. Chakrabarti D, Kumar R, Tomkins A. (2006) Evolutionary clustering. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2006), pp. 554–560.
20. Chi Y, Song X, Zhou D, Hino K, Tseng BL. (2007) Evolutionary spectral clustering by incorporating temporal smoothness In Proceedings of KDD, pp. 153–162.
21. Rosvall M, Bergstrom C. (2010) Mapping change in large networks. PLoS ONE, 5(1), e8694. doi: 10.1371/journal.pone.0008694 PMID: 20111700
22. Greene D, Doyle D, Cunningham P. (2010) Tracking the evolution of communities in dynamic social networks. In Proc. of International Conference on Advances in Social Network Analysis and Mining.
23. Xu KS, Kliger M, Hero AO III. (2014) Adaptive Evolutionary Clustering. Data Mining and Knowledge Discovery, 28(2), 304–336. doi: 10.1007/s10618-012-0302-x
24. Palla G, Barabási AL, Vicsek T, Hungary B. (2007) Quantifying social group evolution, Nature Letters 446. doi: 10.1038/nature05670
25. Lin YR, Chi Y, Zhu S, Sundaram H, Tseng BL. (2009) Analyzing communities and their evolutions in dynamic social networks. ACM Transactions on Knowledge Discovery in Data 3(2).
26. Mucha PJ, Richardson T, Macon K, Porter MA, Onnela JP. (2010) Community structure in time-dependent, multiscale and multiplex networks. Science 328(5980), pp. 876–878. doi: 10.1126/science.1184819 PMID: 20466926
27. Sun J, Faloutsos C, Papadimitriou S, Yu PS. (2007) Graphscope: parameter-free mining of large time-evolving graphs. In Proceedings of the 13th ACM SIGKDD International conference on Knowledge discovery and Data mining (KDD 2007), pp. 687–696.
28. Langone R, Alzate C, Suykens JAK. (2013) Kernel Spectral Clustering with Memory Effect. Physica A 392(10), pp. 2588–2606. doi: 10.1016/j.physa.2013.01.058
29. Langone R, Mall R, Suykens JAK. (2014) Clustering data over time using kernel spectral clustering with memory. In Proceedings of IEEE SSCI CIDM, pp. 1–8.

30. Pratt PJ. (2005) A Guide To SQL, Seventh Edition, Thomson Course Technology, ISBN 978-0-619-21674-0.
31. Jaccard P. (1912) The distribution of flora in the alpine zone. New Phytologist, 11, pp. 37–50. doi: 10.1111/j.1469-8137.1912.tb05611.x
32. Yu CT, Meng W. (1998) Principles of Database Query Processing for Advanced Applications, Morgan Kaufmann, ISBN 978-1-55860-434-6.
33. Murtagh F. (1984) Counting dendograms: A survey, Discrete Applied Mathematics, 7(2), pp. 191–199. doi: 10.1016/0166-218X(84)90066-0
34. Mall R, Langone R, Suykens, JAK. (2014) Agglomerative Hierarchical Kernel Spectral Data Clustering, IEEE Symposium on Computational Intelligence and Data Mining (IEEE SSCI CIDM), pp. 9–16.
35. Lamirel JC, Cuxac P, Mall R, Safi G. (2011) A New Efficient and Unbiased Approach for Clustering Quality Evaluation, PAKDD Workshops, pp. 209–220.
36. Tanahashi Y, Hsueh CH, Ma K. (2015) An Efficient Framework for Generating Storyline Visualizations from Streaming Data, IEEE Transactions on Visualization & Computer Graphics, doi: 10.1109/TVCG.2015.2392771
37. Tanahashi Y and Ma K. (2012) Design Considerations for optimal storyline visualizations, IEEE Transactions on Visualization and Computer Graphics, 18(2), pp. 2679–2688.
38. Eagle N, Pentland AS, Lazer D. (2009) Inferring social network structure using mobile phone data, PNAS, 106(1), pp. 15274–15278. doi: 10.1073/pnas.0900282106 PMID: 19706491
39. Bache K, Lichman M. (2013) UCI Machine Learning Repository, Irvine, CA: University of California, School of Information and Computer Science.
40. Robertson S. (2004) Understanding inverse document frequency: On theoretical arguments for IDF, Journal of Documentation 60(5), pp. 503–520. doi: 10.1108/00220410410560582