Applied Network Science

RESEARCH Open Access

Community-based time segmentation from network snapshots

Streaming and holistic approaches for semi-static and dynamic nodesets

Thomas Magelinski* and Kathleen M. Carley

*Correspondence:
tmagelin@andrew.cmu.edu
CASOS, Institute for Software Research, Carnegie Mellon University, Pittsburgh United States

Abstract
Community detection has proved to be extremely successful in a variety of domains. However, most of the algorithms used in practice assume networks are unchanging in time. This assumption is violated for many datasets, resulting in incorrect or misleading communities. Many different algorithms to rectify this problem have been proposed. Most of them, however, focus on community evolution rather than abrupt changes. The problem of change detection is easier than that of community evolution, and is often sufficient. Here, we propose an algorithm for determining community-based change points from network snapshots. Networks can then be aggregated between change points, and analyzed without violating assumptions. There are three network types that we have defined our algorithm for, each having a case study: static nodesets, semi-static nodesets, and dynamic nodesets. The case studies for these network types are: the Ukrainian Legislature, the Enron email network, and Twitter data from Ukraine. We empirically verify our algorithm in each case study, and compare results to two popular alternatives: Generalized Louvain and GraphScope. We show the impracticality of Generalized Louvain and that our method is less sensitive than GraphScope. Lastly, we use our first two case studies to determine optimal parameters for an anomaly-detection-based streaming method. We then demonstrate that the streaming method was capable of determining events both from data collection errors and from internal network disruptions.

Keywords: Dynamic communities, Community detection, Temporal networks, Streaming data, Verkhovna rada, Twitter networks, Enron

Introduction

While a vast body of literature addresses the problem of community detection in networks, only a small portion considers their dynamic aspects. Recently, this problem has received more attention. The majority of networks are truly time-varying, and community detection should reflect that. Often, large periods of time are aggregated into a static network, or they are aggregated at regular intervals and analyzed individually. At best, this smooths over any interesting temporal features of the network data, at worst, combining links from old and new communities can yield misleading results.

Although infrequently used in practice, there has been work in this area which is well summarized in (Aynaud et al. 2013; Rossetti and Cazabet 2018). Much of this work focuses on community evolution, or how communities change in time (Aggarwal and

Subbian 2014; Greene et al. 2010). We believe that an easier problem has been understudied: We seek to answer the question: "How can we segment a dynamic network, such that static analysis of the segments will be representative of the underlying dynamic communities?" Effectively, temporal partitions should be change points for community structure. This problem is easier in that we assume communities are static until an event changes them noticeably. Segmenting the network in this way gives us access to all the tools of static network science, and simplifies the interpretation of results. "Partitions" are also used to describe the grouping of nodes into communities. In this work, we are describing *temporal* partitions, which define the ends of time segments.

The two main methods of segmenting networks in practice, Generalized Louvain and GraphScope, have some shortcomings. The first is their dependence on a specific grouping algorithm. As is well known, each grouping algorithm has strengths and weaknesses, and should not be used a universal tool. Generalized Louvain has the disadvantages of relying on user-defined parameters, and needing a static nodeset. GraphScope is parameter free, but relies on information-theoretic clustering and is not robust to noisy networks. Through two case-studies, we demonstrate the effects of these shortcomings in practice.

In this work, we propose a simple method for placing temporal partitions in dynamic networks such that static community analysis accurately represents the dynamic communities. This method is parameter free, works with any grouping algorithm, and is somewhat robust to noise, as demonstrated through trials with synthetic datasets and two case studies.

The initial framing of the method relies on static nodesets, where the Ukrainian Parliamentary voting network is analyzed as a case-study. This assumption is relaxed for semi-static nodesets, and the Enron email database is studied. Finally, a streaming alternative is introduced and optimized to achieve similar results to the first two case studies. It is then applied to Twitter data streamed from Ukraine.

Current tools are typically only defined for one or two of these classes of nodesets. As such, we will only compare to methods that are applicable. Generalized Louvain is most appropriate for static nodesets, so is compared to our work in the "Comparison to Generalized Louvain" section. GraphScope, is most appropriate for semi-static or dynamic nodesets, so is compared to our work in the "Comparison to GraphScope" section. We do not compare to GraphScope for our streaming work, though it is applicable. The Twitter application is shown as a proof of concept; many case studies would need to be performed to fit parameters with generalizability. However, given the empirical validation of the method with our two case studies, we feel this method is a promising avenue for future work.

Prior work

The problem of community detection in complex networks has received a tremendous amount of attention, resulting in many popular algorithms that have been empirically verified (Blondel et al. 2008; Newman 2004; Ng et al. 2002). However, these algorithms all assume that the network being analyzed is *static*. If this assumption is violated, different communities may have been averaged together over time, resulting in obscured or misleading results. While dynamic aspects of communities are still often ignored in practice, many potential methods of dynamic community detection have been proposed. Rossetti and Cazabet posit that this is due to a disconnect between researchers in the field, and a lack of visibility (Rossetti and Cazabet 2018). Here, we discuss how prior work in dynamic community detection has motivated our approach.

Using the terminology of Rossetti and Cazabet, there are two types of dynamic network models: snapshots, and temporal networks. Snapshots segment the data into networks that are assumed to be static, while temporal networks assign a birth and death time-stamp to each edge. The temporal network model is pure in that it does not aggregate links, and the network is never assumed to be static. While this is more accurate than the snapshot approach, it comes with limitations. Namely, the analysis for such objects require more computational power, and a set of tools separate from those created for static networks. Network snapshots, however, have access to the large toolset of network science. Furthermore, it is critical that community detection can extend to streaming data. Snapshots are a natural way of handling this: aggregate links in the stream until the snapshot length has been reached, then analyze it. This approach has been used by one of the most popular streaming methods in this space, GraphScope (Sun et al. 2007). Currently, the length of a time slice is best chosen according to the rule of thumb: “slices should be 5 times shorter than the scale of interest.” That is, if week-level changes were of interest to a researcher, slice length of one day might be appropriate.

Snapshots are limited, however, when the goal is to find fine-grained evolutions in a network. In this case, temporal networks are more appropriate. Here, we are looking for *events*, or large changes in communities, rather than evolution patterns, so we have chosen to use the snapshot modeling approach. Given this, our partitioning techniques will work best under the assumption that network communities undergo rapid change, meaning in few time slices, rather than communities undergoing constant structural evolution. This assumption is often met when major events occur in the network’s timeline.

Using network snapshots, it is common to take an “instant optimal” or a “two-step” approach, wherein snapshots are grouped statically and then compared (Aynaud et al. 2013; Rossetti and Cazabet 2018). Some others have criticized this approach, claiming that it is too vulnerable to noise and the snapshot groups do not use valuable historical data (Lin et al. 2008; 2009). These concerns have merit. It has been shown that static grouping algorithms are unstable, and can give very different result under only small perturbations to a network (Aynaud and Guillaume 2010). Given that slice groupings are expected to be noisy, we compute pairwise comparison for *all* time slices. Comparing all slices addresses both of the concerns voiced in (Lin et al. 2008; 2009); historical data is used when finding similar segments, and pairwise-noise will be present, but should average out when considering an entire block of similarities. It seems that only Goldberg et al. have used all slices in the comparison step of a two-step approach (Goldberg et al. 2011). Our work differs from Goldberg et al.’s in two ways. First, our goals are different. They sought to identify evolutionary patterns in groups, while we aim to find disruptive events with respect to communities, in hopes of aggregating network snapshots into a smaller series of networks with meaningful divisions. Second, our approaches differ. They identified common community cores across time, while we calculate the overall correlation between communities.

A very similar approach is given by Masuda and Holme, who use hierarchical clustering to label slices as “states” of the system, which are expected to recur (Masuda and Holme 2019). Their work differs from ours in two major ways. First, they take a “one step” approach, where states are decided based on the network rather than its communities.

Second, no temporal continuity is imposed for states. For our purposes, this is essential. Without temporal continuity, states cannot be collapsed and analyzed as a static network. Also, we rely on a different comparison mechanism: product-moment correlation. The fact that pairwise temporal similarity operations find success in network aggregation (our work) and chain-like state changes (Masuda and Holme), shows the power of the approach for solving new problems in temporal networks.

Our approach is a special form of link aggregation. The problem of link aggregation has been studied in (Taylor et al. 2017). Taylor, Caceres, and Mucha examine the effect that aggregation has on community detection. They look at both aggregation over network modes and over time slices. It was concluded that aggregation can enhance or obscure communities depending on their size and persistence. Matias and Miele recognize a similar problem, questioning the assumption that most nodes do not change groups (Matias and Miele 2017). While Taylor et al suggest analysis on multiple scales and Matias and Miele attempt to control for short term group-switching, we take a different approach: only aggregate slices that have similar community structure.

While much of network analysis ignores group dynamics, analysis considering group dynamics often uses one of two tools: Generalized Louvain or GraphScope (Mucha et al. 2010; Sun et al. 2007). As such, we will demonstrate our results through comparison to these two methods in the "Case study: Ukrainian Parliament" and "Case study: Enron emails" sections, respectively. Mucha et al. have generalized the original Louvain grouping algorithm from unipartite to multipartite graphs. Network snapshots, then, are considered an ordered multipartite graph. This algorithm uses two resolution parameters. We show that tuning these parameters in practice is very difficult, and still leads to imperfect results. GraphScope is a streaming algorithm, meaning that it takes in snapshots sequentially. It groups each snapshot and calculates whether or not it should be included with the previous time slice, all using the concept of minimal descriptive length. As previously stated, streaming approaches are necessary for analyzing certain datasets, such as those coming from social media. However, GraphScope's biggest weakness is that it relies on information-theoretic clustering, which is not as well-verified as other common methods. We show that GraphScope is also likely to be too sensitive in detecting changes. To rectify these problems, we modify our approach to support streaming in the "Dynamic nodesets: a streaming approximation" section. Instead of information theory, we draw on basic principles of anomaly detection, which has been done by others, such as in (Li et al. 2017; McCulloh and Carley 2011; Priebe et al. 2005). A similar streaming method has been proposed by Morini et al (Morini et al. 2017). However, their work is more similar to Goldberg et al.'s. Further differences will be discussed in the "Dynamic nodesets: a streaming approximation" section.

Other recently developed streaming or online techniques have also analyzed the Enron dataset (Miller and Mokryn 2018; Peel and Clauset 2015). Peel and Clauset couple the generalized hierarchical random graph model with Bayesian hypothesis testing to find breaks. Miller and Mokryn argue than analyze networks or communities directly, the degree distribution can be used to find structural breaks. Both methods only analyze the 151 management employees in the Enron dataset, so will not be used for comparison here.

In summary, there is a need for simplistic community-based event detection in network science. Work has been done in this space, specifically using network snapshots, but focuses on community evolution patterns, rather than shocks to the overall structure.

In the following sections, we outline a method for detecting such events, and compare results to two popular algorithms: Generalized Louvain and GraphScope, showing benefits of our method for each case.

Static nodeset

Methods

Informally, the goal of the work is to partition the time dimension of a network such that each segment forms *cohesive groups*, and adjacent segments are *noticeably different* from each other. This section will formalize a procedure that achieves this goal.

We will begin by taking an offline, or non-streaming approach. While streaming approaches have advantages, as discussed in the "Dynamic nodesets: a streaming approximation" section, many datasets are analyzed after they are collected in full. Streaming has no knowledge of "future" data, and non-streaming approaches can leverage all of information available to obtain better results. The most notable disadvantage of many streaming approaches is the inability to determine whether abrupt changes in incoming data is indicative of a real change or a relatively short burst of noise.

Additionally, we start with a fairly restrictive assumption: the nodeset is static. That is, every node is present in every time slice. While many datasets violate this assumption, this is a natural starting point. Dynamic nodesets introduce complications such as the problem of comparing groups with different nodes, which are further investigated in the "Semi-static nodesets" section.

Each community detection algorithm has strengths and weaknesses, and is thus more or less suitable for certain types of networks. For example, Louvain grouping has proved to be extremely successful, but is ill-defined for dense weighted networks. Further a "no free lunch theorem" for community detection networks has been proven, stating that no algorithm is optimal for all community structures (Peel et al. 2017). As such, a temporal partitioning procedures which are independent of grouping algorithm are preferred over those that rely on a specific method, such as Generalized Louvain (Mucha et al. 2010).

A partitioning method independent of community detection algorithm can be constructed using the co-group network. That is, the network between nodes where links represent shared group membership. After calculating the co-group matrices, we will compare them to find natural partitions in time. This way, the user can choose any algorithm they like to group the slices before the temporal analysis begins. It is important to note that comparing co-group networks relies heavily on the static nodeset assumption. One way of comparing two networks is through the Product-Moment Correlation (Krackhardt 1987). The correlation, ρ, between two co-group matrices A, and B is given by:

$$\rho(A,B) = \frac{cov(A,B)}{\sqrt{cov(A,A)cov(B,B)}} \tag{1}$$

$$cov(A,B) = \sum_{i,j}(A_{i,j} - \mu_A)(B_{i,j} - \mu_B). \tag{2}$$

Throughout the work we define $A_{i,i} = 1$ in co-group matrices, to indicate that a node is always in its own group.

We take what is known as a "snapshot" approach to dynamic networks in the language of Rossetti and Cazabet. That is, the initial temporal network can be defined as a series of adjacency matrices, $A_1, ...A_T$, at every time step, $1, ..., T$. After these are all grouped there is a series of co-group matrices, $CG_1, ..., CG_T$. Pairwise similarity is calculated between all of these slices to obtain the similarity matrix, S:

$$S_{t_1,t_2} = \rho\left(CG_{t_1}, CG_{t_2}\right). \tag{3}$$

Under the assumption that community change occurs rapidly, S will have near block-diagonal structure. Communities will remain roughly unchanged between events, causing diagonal blocks in S with high similarity. After changes, two segments will be fairly different, resulting in low similarity in S's off-diagonal blocks. We now formally define a method for locating each block's boundaries, which correspond to community change points.

Now that similarity is defined for our temporal network, we can formalize the goal of "cohesive groups" as *high internal similarity*. For example, if a time segment begins at $t = 10$ and ends at $t = 20$, the values within the square similarity matrix $S_{10:20,10:20}$ should be high. As such, we define the time partitioning problem to maximize this term. The general temporal partitioning problem is to find a list, $\mathbf{b}$, which contains start and stop points, or boundaries, of time segments. We will assume there are P partitions, and $\mathbf{b}$ is strictly increasing with fixed ends $\mathbf{b}_1 = 1$ and $\mathbf{b}_P = T + 1$, so that the partitions are well defined. Then, the problem is stated as:

$$\arg\max_{\mathbf{b}} s_{internal}(S, \mathbf{b}) \tag{4}$$

$$s_{internal}(S, \mathbf{b}) = \frac{1}{n_i} \sum_{k=1}^{P-1} \sum_{i=b_k}^{b_{k+1}-1} \sum_{j=b_k}^{b_{k+1}-1} S_{i,j}, \tag{5}$$

where n_i is the number of entries in all the internal blocks, and i, j are the indices of the sub-matrices. Given P, this problem can be solved quickly, especially considering that P will typically be small in practice. This could potentially be written as a dynamic program, though with the scale of data used here such an improvement is unnecessary.

However, P should not have to be given. Domain knowledge might give a reasonable guess as to what P should be, but there is no telling whether or not the structure of groups would have actually changed due to outside events. There could also be unknown events.

A natural criteria for determining P comes from the second stated goal: find adjacent segments which are noticeably different from each other. Now, P can start at its minimal possible value, 3, and be iteratively increased to meet both goals. Just as segment similarity is encoded in S, so is segment difference. Using the example from before, where a segment begins at $t = 10$ and ends at $t = 20$, add the fact that the next segment ends at $t = 30$. If these segments are very different the values within $S_{10:20,20:30}$ should be low. Calculating this difference for a full time segment with length P, the external similarity is:

$$s_{external}(S, \mathbf{b}) = \frac{1}{n_e} \sum_{k=1}^{P-2} \sum_{i=b_k}^{b_{k+1}-1} \sum_{j=b_{k+1}}^{b_{k+2}-1} S_{i,j}, \tag{6}$$

where n_e is the number of entries in all the external blocks, and i, j are the indices of the similarity matrix, and k indexes the boundary list $\mathbf{b}$. Naturally, there is a tradeoff between these values. We can expect the internal similarity to increase even beyond the optimal partitions, albeit more slowly. This happens by splitting already cohesive sections into more cohesive segments. When this happens, the external similarity should begin to increase. Since similarities will be calculated iteratively, a subscript will be used to denote which iteration the similarity was calculated on. For example, s^1_{internal} is the initial internal similarity.

Increasing external similarity is a sign of over-partitioning, so is naturally a good stopping criteria. However, we cannot expect real-world data to exhibit perfect block-diagonal structure. Thus, we propose a stopping criteria based on the rate at which each similarity changes: $\Delta s^t_{\text{internal}}$, $\Delta s^t_{\text{external}}$:

$$\Delta s^t_{\text{internal}} = \frac{s^t_{\text{internal}} - s^{t-1}_{\text{internal}}}{s^1_{\text{internal}}}, \tag{7}$$

replacing "internal" with "external" yields the equation for $\Delta s^t_{\text{external}}$. If $\Delta s_{\text{internal}} > -1 * \Delta s_{\text{external}}$, too many partitions have been placed. The negative sign is due to the expectation that $\Delta s_{\text{external}}$ is negative. Thus, the final algorithm is:

1. Initialize $P = 3, t = 1$
2. Select partition list $\mathbf{b}^t$ of length P that satisfies the partition requirements and maximizes internal similarity
3. Calculate $s^t_{\text{internal}}, s^t_{\text{external}}$
4. $P = P + 1, t = t + 1$.
5. Select partition list $\mathbf{b}^t$ of length P that satisfies the partition requirements and maximizes internal similarity
6. Calculate $s^t_{\text{internal}}, s^t_{\text{external}}$
7. Calculate $\Delta s^t_{\text{internal}}, \Delta s^t_{\text{external}}$
8. If $\Delta s_{\text{internal}} < -1 * \Delta s_{\text{external}}$: return to step 4.
   Else: final partitions are given by $\mathbf{b}^{t-1}$.

This two-step procedure first ensures that the segments are as similar as possible, and then stops when increasing P fails to yield additional segments that are meaningfully different. Combining the two goals into a single objective function sometimes led to inaccurate partitions, since segments were blended in order obtain low external similarity.

The algorithm could be forced to continue, which will give more breakpoints. These could also be interesting but it should be remembered that they are not as significant as the initial break-points.

Testing on synthetic datasets

Network datasets with ground truth communities are rare. Rarer still, are network datasets with ground-truth community disruption. Therefore, we test the validity of our approach using a series of experiments on synthetic datasets. With synthetic datasets, we can impose change points and access our ability to recover them. Throughout these experiments we chose a nodeset size of 500. Additionally, we are only considering Erdös-Rényi random networks of varying density between experiments. Random networks typically have low modularity, and as such are a good test case. Further, Peel et al. have concluded that verification on embedded communities is flawed, so we do not rely on manually embedded communities here (Peel et al. 2017). If our algorithm can detect changes in weak communities, it should perform well in the easier case of strong communities. For all tests, a temporal network with 20 slices was considered. The ground-truth breaks were placed at $t = 4, 8, 17$. Every experiment was repeated 100 times.

First, a very basic set of tests were performed. At time $t = 0$, a random network was constructed. Then, each slice up until the first break was set to this network. When a break occurs, a new random network was generated and the process repeats. Basically, slices are identical within breaks, but completely different random networks between breaks. In this case, then, the ideal internal similarity score is 1, and the external similarity will likely be close to 0. This construction method was tested with density 0.1, 0.2, and 0.5. For each experiment the algorithm gave the exact set of breaks for all 100 repetitions.

Second, a more realistic set of tests were performed. Again, a random network was generated and slices were set to that network up until a break. This time, when a break occurred some fraction, f, of the links were randomly rewired. In these experiments, internal similarity is still 1, but the network is retaining some of its original structure throughout the timeline, giving higher external similarity. For these experiments, density was held to a constant 0.1, but f was varied: $f \in [0.5, 0.1, 0.05, 0.01]$, to measure how the algorithm performs on changes of varying scales. For $f \in [0.5, 0.05]$ the exact partitions were recovered in all 100 trials. For $f \in [0.1, 0.01]$, the partitions were recovered in 99 trials. In the remaining trial the break at $t = 4$ was not detected, while the other breaks were. As f decreases, it is increasingly likely that the underlying communities do not change, so it is expected in some instances we will not see breaks. Given our results up to changes in only 1% of links, it seems that our method is extremely accurate in a noiseless environment.

Lastly, two additional experiments were conducted to study the effect of noise. Now, since the networks in question have low modularity, random changes in links can potentially have a large impact in group structure. To test this we followed the same procedure as in the second set of experiments, but adding the additional step of swapping some fraction, n, of the links *at each time slice*. In these experiments we held density to a constant 0.1, f to a constant 0.5, and tested n values of 0.01 and 0.05. The exact partitions were recovered in 88 and 66 trials, for 1% and 5% rewiring, respectively. Typically, the algorithm had only one of the following errors: one of the partitions was misplaced by 1 slice, one of the partitions was missing, or an additional partition was added.

Given the extremity of the experimental conditions (testing on networks with low community structure and high sensitivity to noise), and the results (>99% accuracy in noiseless scenario, >65% accuracy under extreme noise), these results bolster the method's validity.

Case study: Ukrainian Parliament

It is known that the Ukrainian Parliament, the Verkhovna Rada, has interesting political groups within it called “factions” (Duncan 1958). Factions are interesting as they extend beyond party boundaries and change dynamically. Prior work shows that factions can be obtained from network analysis of voting data (Poole 2005; Poole and Rosenthal 1985). Further, there is known to be a large disruption of alliances in convocation 7, spanning from 2012 to 2014, in which a revolution took place.

Thus, in this case study we analyze the Rada voting data from convocation 7, which is available publicly (Verkhovna Rada of Ukraine 2018). The dataset from this convocation analyzed included 493 bills, over 91 time slices. Time slices are defined using the day in which bills are registered. Some time slices may have multiple bills, some may not. There are six voting options in the Rada: for, against, did not vote, no vote, absence, and abstain. Domain experts have suggested that votes other than for and against are all used to mean the same thing: they are not in favor of the bill, but do not want to send a strong signal against it. As such, these votes are not considered as ties between MPs. The voting network, then, is the network constructed so that nodes are parliamentarians and the weighted links are the instances of co-voting between two parliamentarians in the given time period.

One method of determining factions from this network is using Louvain grouping. Thus, we use Louvain grouping at each of the 91 slices, and obtain the similarity matrix. Note that this procedure can lead to a significant number of isolates at each time slice.

It was determined that there is only 1 partition in the networks timeline, occurring on February 6, 2014. Figure 1 shows the similarity matrix with the induced temporal partitions. It can be seen that the "internal similarity" of the partitions is high, often around 0.8, and the "external similarity is low, often around 0; specifically, the average value internal similarity is 0.25, and the average external similarity is 0.03. While there are many entries in *S* with low similarity during segment 2, they are not pervasive. Meaning, since the overall block has high similarity, these intermittent entries of lower similarity are normal variations in the communities. If a meaningful change was occurring, slices would not be highly similar to entries within the block on average, i.e. a new diagonal block would form in the matrix.

The Ukrainian revolution started in February of 2014, culminating in the overthrow of the Ukrainian government and the removal of President Victor Yanukovych. As expected, our algorithm returns this as the most significant change point. Forcing our partitioning to continue revealed another date: May 15, 2014. This is interesting in that it is the last slice before the presidential election, occurring on May 25. Again, groups are highly correlated before and after these events, so there was not much of an impact on the communities overall.

Now that the temporal network has been segmented, we can analyze the resulting two static networks statically. First, we visualize the network in Fig. 2. This figure shows that the initial two communities within the Rada relied on only a few parliamentarians to bridge the gap between them. Then, we see that after the event, the community

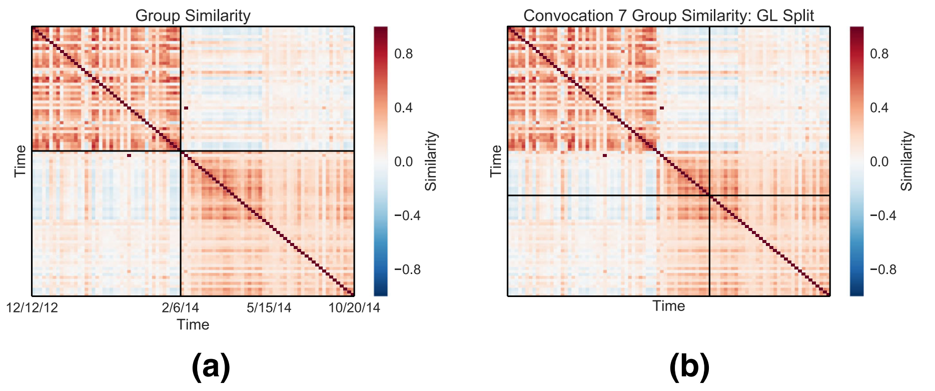

Fig. 1 The Rada's temporal similarity matrix for convocation 7, created using Louvain grouping. The temporal partition found with our method is drawn in black in (**a**), and occurs on February 6, 2014, which is the start date of the Ukrainian Revolution. The best-fit generalized Louvain partition is shown in (**b**), and occurs on April 4, 2014

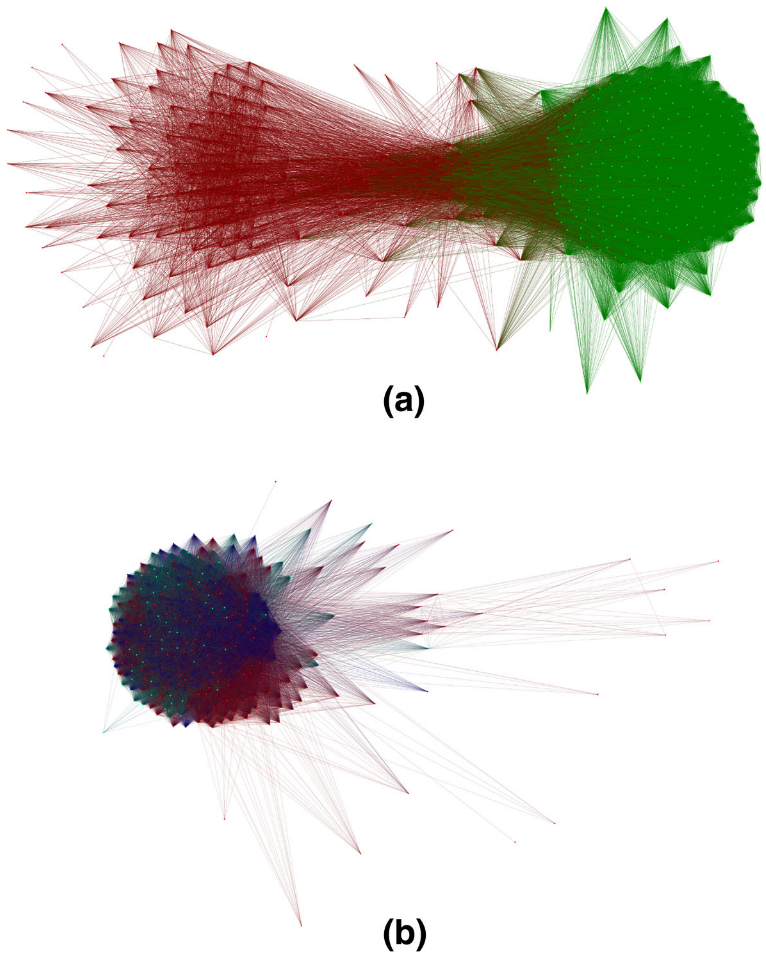

Fig. 2 Network visualization of the Rada in the first time segment in (**a**) and the second in (**b**). Links below the mean link value are not shown, nodes are colored by Louvain grouping for the individual time segment. The initial time segment shows clear group structure in the first time segment, with a minority group detached from a central core. The second time segment does not have clear groups, because the groups found are all inter-related

structure cannot be discerned visually, indicating poor grouping after the event. This result is confirmed quantitatively using Louvain modularity; the initial time segment had modularity 0.139 while the second only had 0.024.

To better understand how the groups have changed, a Sankey Diagram is displayed in Fig. 3. We see that group A transitions from holding a majority to holding only a third of the seats. This is due to a large number of its constituents joining members from group B to form a third group, and a smaller number of constituents joining the opposing group. Both before and after there is a small number of MP's failing to cast a significant vote in each time period. Here, significant refers to the fact that "co-voting" relies on non-abstention, and some MP's strictly cast abstention-type votes.

Comparison to Generalized Louvain

Generalized Louvain requires two resolution parameters: ω, γ. Currently, there is no way of objectively selecting these parameters (Mucha et al. 2010). So, we perform a grid search over the parameters suggested in their initial work, adding additional values making the

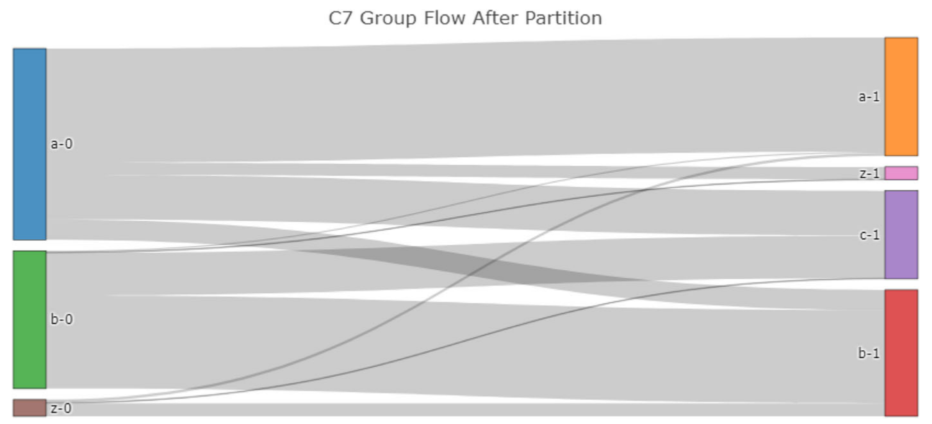

Fig. 3 Sankey visualization of the flow from Rada groups in time segment 1 to time segment 2. Groups in the "z" category are isolate nodes that were lumped together for convenience. The main finding here is the birth of a third group, which drains the controlling faction of its majority, giving the opposing faction the most power after the revolution

total space: $\omega \in [0.25, 0.5, ..., 4, 5, 6, ..., 10]$, $\gamma \in [0, 0.1, 1, 1.5, 2, ..., 6]$. Out of these 286 possible combinations, only 14 partitioned the data less than 10 times. Finally, one partition was best in terms of all score metrics (internal similarity, external, ratio, difference), which was obtained from $\gamma = 2.5, \omega = 8$. This parameter combination yields one partition in the voting data, on April 4, 2018. This partition is visualized in Fig. 1. Clearly, this result is sub-optimal.

Case study conclusion

This example shows why dynamic community detection is necessary: aggregating the entire convocation 7 data into one network would have lost the two very different behaviors seen in the data. Our methodology partitions the temporal network at exactly the point that the legislature Ukrainian revolution began, giving empirical validation to our results. Additionally, allowing our algorithm to over-partition the data revealed two other change points, centering around the election. Finally, we show that performing Generalized Louvain with 286 parameter combinations led to only 14 usable partitions, all still with sub-optimal results. Additionally, the results from Generalized Louvain could not easily be compared without introducing some similarity measures, as we have done here.

Semi-static nodesets

Methods

As stated, the above method of finding optimal time segments requires that nodes be present in each time slices. However, many datasets do not meet this requirement. In some datasets, such as the Enron email corpus, key actors are present in *most* of the time slices. In this section we will make changes to the initial methodology that allow nodes to be missing in some slices.

Again, each time-slice is grouped, and the co-group matrix is calculated. However, now that nodes in our nodeset can be "out of the network" completely, isolate nodes should not be allowed. Instead isolate nodes will be entered as "not a number" in the co-group matrix. Then, the correlation function is adjusted to ignore these values in calculation. Basically, the correlation now is calculating the similarity between groups of nodes present in both time slices.

Additionally, an added assumption can fill in some missing data: nodes do not change group affiliations in slices they are not present for. That is, if a node is not present until $t = 4$, is present in $t = 5$, and then is not present again, it will have "not a number" co-groupings until time $t = 5$, then it will retain the time $t = 5$ co-group ties for the rest of the dataset. To be clear, the assumption is that nodes only change group through forming other links, or the lack of a link cannot be used to change a node's group. Theoretically, this seems like reasonable way to fill in missing data. In practice, however, this added assumption actually blends the time slice networks together, making it harder to establish temporal partitions. Thus, this assumption is **not** made in this work.

Now, nodes can enter and leave the dataset. However, large sets of infrequent nodes can disrupt results. For example, if many nodes are present in an early slice, but never return, it is possible that subsequent slices appear more correlated than they should. Issues like this can be resolved in two steps. The first part of this issue stems from the concept of time slices itself. What is a time slice? It depends on the dataset, but often it is up to the user to define a sensible time slice. Slices are arbitrarily selected as regular intervals like days, weeks, or months. Ideally, the length of the time slices should be 4–10 times shorter than that which meaningful change might occur for that network. Given the somewhat arbitrary nature of the current slice length choice practices, we suggest that the node frequency should also be taken into account. Again considering an email network, it is not reasonable that everyone emails every single day, but most people send at least an email a week, so this may be a better time slice. A month would include even more users over a frequency minimum, but offers less resolution for temporal changes. These trade-offs must be looked at on a case-by-case basis.

In the case of semi-static nodesets, many nodes in the network may appear infrequently. If a researcher would like to answer the question "how do *core* members of this network change their community structure?" Infrequent nodes may want to be filtered out. This is not a necessary step, as nodes will only have an impact on results when they *are present*, however, large numbers of less important nodes can obscure results and make analysis more challenging. As such, a node frequency threshold can be introduced. After the network slices are created, define n_i as the number of slices node i is present for. Then, a define λ as a threshold such that all nodes with $n_i < \lambda$ are removed from the analysis. For example, $\lambda = \frac{3}{4}T$ retains nodes that are present three quarters of the time. This node-filtering step is prevents large numbers of sporadic nodes from skewing the analysis of our segments.

This is not a parameter to be tuned, or adjusted to get better results. Rather, it is an optional pre-processing step which may be helpful for researches interested in the evolution of nodes that are frequently present. In the Enron example, we are only interested in how communities change around users closely associated with the organization. If there is an outside user who is emailed 3 times over the 2 and a half year dataset, they are not relevant.

Case study: Enron emails

The Enron dataset has been a case study for an extensive amount of work on community detection, including the aforementioned work on GraphScope (http://www.cs.cmu.edu/enron/) (Rowe et al. 2007; Sun et al. 2007). We use the dataset from the company's launch in November 1999 to May 2002. It contains approximately 1.6M emails. Our purposes for using the data are twofold. First, it is a commonly used example of a dynamic network with a semi-static nodeset. Second, we can then directly compare it with GraphScope's results.

Since the email network is large and fairly sparse, Louvain grouping is again appropriate. To fairly compare with GraphScope, we use a week-level time slice, and consider the data between July 10, 2000 and November 11, 2001, for a total of 70 time slices. Examining the node frequency distribution showed that an overwhelming number of the email accounts were only present for 3 or less weeks, 76.1% to be precise. Still, we set a threshold of $\lambda = \frac{1}{2}T = 35$, which drops the number of nodes from 60,637 to only 1672. This large drop shows why the threshold is necessary in the first place. Without any filtering, 97.2% of the nodes in the dataset would have nearly-unchanging group affiliations because they aren't reappearing in the data. Unfortunately, filtering nodes based on their frequency changes the dataset significantly from its form when analyzed by GraphScope. It might be interesting to see the dropped node's group structure, but since they are so infrequent that is no longer a dynamic question. Filtering down to the reoccurring nodes allows for an unobscured view of the dynamic communities, so we proceed despite the differences to GraphScope.

A single dynamic partition for this dataset was found on June 18, 2001. The partition in the similarity matrix is visualized in Fig. 4. Again, our partition is intuitive, providing high internal similarity in both segments while maintaining low external similarity. Other events can be seen on December 18, 2000, and December 10, 2001, however, the communities before and after these events are still highly correlated, so the change was not large. The major change occurs as Rove divested his stocks in energy, and just before Kenneth Lay took over as CEO. Both of these changes were seen using GraphScope.

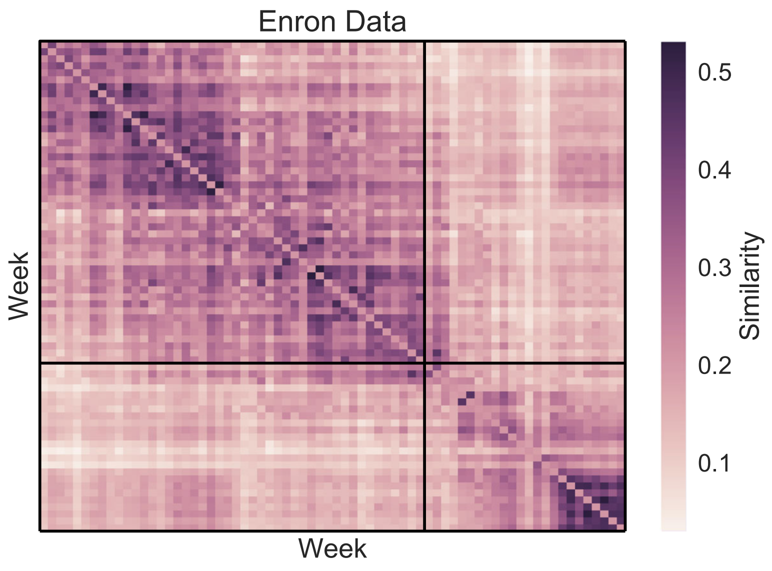

Fig. 4 The Enron temporal similarity matrix. Our partition is shown in black on June 18, 2001. Note that the diagonal has been artificially set to the mean similarity for visualization purposes, where in reality it is equal to one

Comparison to GraphScope

The biggest difference between our results and results obtained using GraphScope is in the number of temporal partitions. We found 1 partition that showed very different community structure given recurring nodes, while they found about 15. It seems from our analysis that GraphScope is more sensitive to small changes, since the changes they find lead to highly correlated time segments in our analysis.

Another potential explanation is due to the fundamental differences in the algorithms. It is possible that the streaming approach taken by GraphScope can partition the data based on large changes in the nodeset, and it is likely that streaming approaches give more partitions in general. Given the restriction on the data that they have access to, quick bursts of noise may appear to be events. These bursts are desirable in some contexts, seeing that they occur around major events. However, if the groups quickly normalize, those interested in community change points may need a more coarse definition of changes.

Case study conclusion

From our analysis, only one major change in the Enron email communities occurred over the duration of the dataset. This change occurred between two events also noted by GraphScope: Rove divesting his stocks in energy and Kenneth Lay taking over as CEO. While we do find change points at many major external events, as GraphScope does, we believe this is due to non-community related changes. From our analysis, though these changes may be important, they are not reflected in the community structure alone, and thus cannot be detected here.

Dynamic nodesets: a streaming approximation

While there are plenty of examples of temporal networks with nearly-static nodesets, there are also many examples where node presence is very inconsistent. Perhaps most notably, social media data such as Twitter data has this property. If our line of reasoning is to be applied to these datasets with ever-increasing importance, we need to adjust our methods accordingly. To do this, we take a streaming approach.

For real-time analysis of large social media data, only a streaming approach is practical. However, a streaming approach will always be worse than a holistic data approach, since it is unable to take advantage of future data. Given this, we use our prior results for the Rada and Enron as target results for our streaming algorithm, and tune its parameters to match as closely as possible. A streaming approach similar in nature is given by Morini et al. in (Morini et al. 2017), but they lack quantitative validation and empirical validation through a non-streaming approach.

Again, the purpose of this section is to show that parameter tuning of such a model is *possible*. With only 2 case studies to tune parameters, it is not possible to obtain a truly general model. However, with tuning from only these 2 studies we find interesting change points from real social media data.

Methods

In areas where streaming is required, speed of computation is hugely important. A major bottleneck in our prior algorithms for dynamic partitions is the slice-level grouping computation. Grouping algorithms are expensive, and while our procedure currently is more efficient than procedures requiring multiple groupings per slice, such as GraphScope, limiting the number of calls to such algorithms can give great payoffs. Thus, we explore how different our segments would be if we did not group the time slices at all, but instead calculated the correlation of adjacency matrices directly.

In one sense, this method gives *more* insight to changes in the network, because small changes are not averaged away through group assignments. However, now detected changes might be based on inconsequential links which do not affect the community structure. To quantify this effect, we compute the correlation of the output matrices. That is, one temporal similarity matrix was created based on co-group slices (as before), and one was calculated based on the adjacency slices. The correlation between these matrices was 0.81 for the Enron network, and 0.84 for the Rada. Given these high correlations, we proceed with the adjacency similarity for the streaming approach.

Anomaly detection is easiest to perform on one signal. As such, we seek to combine our initial objectives into a single equation. Initially, we were trying to maximize internal similarity while minimizing external similarity. Thus, the initial objective function was closely related to the function:

$$F(S, \mathbf{b}) = s_{internal}(S, \mathbf{b}) - s_{external}(S, \mathbf{b}), \tag{8}$$

where S was the temporal similarity matrix over the whole timeline, and $\mathbf{b}$ is the list containing segment boundaries.

In a streaming setting, this function is problematic. As time goes on, the timeline expands, so S increases in size. This results in computations that are more and more expensive. This can be resolved by considering only a fixed window size. If a window of size w is used, only the latest w slices are used in constructing S. While resolving the increasing complexity issue, a windowed similarity matrix is only an approximation of the full matrix. Still, the values within this matrix will be indicators of recent change in community structure. Thus, by detecting anomalies in F, we may recover the change points found using the offline approach.

Previously, $\mathbf{b}$ was found through the algorithmic process detailed in the "Methods" section. In a streaming setting, however, the goal is not to place partitions, but to determine if a break has just occurred. So, here, $\mathbf{b}$, is predefined based on the window size. This way, F only changes through updates in the similarity matrix, S.

The smaller the window that F is calculated at, the more precisely a partition can be placed. Larger windows will give more accurate estimates of the objective, but are harder to compute with large highly dynamic nodesets, and make partitioning less precise. We proceed with the smallest possible window size, 3. If the window was smaller than 3, the similarity matrix could not be partitioned to have meaningful internal similarity. There are two ways to partition a matrix of size 3: after the first slice or after the second slice. We chose the latter so that anomalies can be detected at the most recent slice.

For our choice of window and partition, the objective function can be defined:

$$G = S_{1,2} - \frac{S_{1,3} + S_{2,3}}{2}, \tag{9}$$

where S is the temporal similarity matrix of the most recent 3 time slices. If a large change occurs the previous two time slices will have high similarity to each other and low similarity to the new slices, yielding a large value of G.

It is standard practice in anomaly detection to estimate the running mean and variance of a signal, and report an anomaly if the signal exceeds s standard deviations of the signal for n consecutive time stamps. As such, we test four parameters: n time stamps, s_h standard deviations above the mean, and s_l standard deviations below the mean, and ℓ, the number of values required to start detection. We grid search over the space $s_l, s_h \in [0, 0.1, 0.25, 0.5, 1, 1.5, 2, 2.5, 3]$, $n \in [1, 2, 3, 4]$, $\ell \in [3, 4, 5, 6]$, to find the optimal parameters for our two previous case studies. The temporal partitions were evaluated using the same function F, but on the whole similarity matrix. The highest normalized sum of objective over both domains was selected.

The optimal parameters were determined to be $s_l = 2, s_h = 3, n = 1, \ell = 5$. Effectively, this means that the most effective streaming approach made partitions when a single large deviation occurred, provided that there was at least 5 points in the series. Using the adjacency matrices and these parameters resulted in the same partition found for the Rada previously, and Enron partitions on December 18, 2000, February 5, 2001, and June 18, 2001. Basically, this obtained the optimal Enron partition, plus two extra partitions, one occurring when Jeffery Skilling took over as CEO. Since this approximation gives similar results and is significantly more efficient, we use it moving forward.

Case study: Ukrainian Twitter data

We tested this streaming method on Twitter data streamed from Twitter's API, placing a bounding box over Ukraine. The data was collected from January 5, 2018 until June 9, 2018. The filtered data had a total of 1,646,446 tweets, and 169,699 unique users over 112 day-long snapshots after filtering. Due to a collection error, the month of March is missing. The data was streamed normally to test how the algorithm handled such problems.

This resulted in 4 temporal partitions: April 6, April 13, May 16, and May 23. Basic statistics for each segment are shown in Table 1. Note that the first partition occurs on April 6, two network snapshots after the collection error. This verifies that our algorithm can handle "built-in" change points due to errors or some other internal source. The algorithm segments the timeline into 2 short, dense segments, and 3 longer more sparse segments. Hashtag analysis shows that the conversation in each time segment revolves around the K-Pop Band, BTS. Segments then seem to give campaigns for different music award shows: iHeartMusic Awards, MTV Miaw, Billboard Music Awards, and MTV Miaw again. The last segment also contains discussion around the UEFA Champions League soccer final, which took place in Kiev. It is possible that this is the product of bot activity programmed to support the super-group, but it is also possible that these are real users. The top hashtag in the longest segment only occurred 400 times per day, so it could be from a relatively small but dedicated following.

Table 1 Summary statistics for the each Twitter segment

Start Date	End Date	Days	Tweets per Day	Users per Day	Density
January 5	April 6	48	2505.2	1765.5	1.67e-5
April 7	April 13	6	3345.7	3127.2	5.70e-5
April 13	May 16	33	3045.0	2192.4	1.92e-5
May 17	May 23	7	5383.6	3825.3	5.26e-5
May 23	June 9	17	4122.1	3039.6	2.62e-5

Note that "day" normalization is days *in the dataset*. We noted that a collection error led to no tweets in March, the day count used is less than the days from the beginning to the end. Density refers to the density of the network created between tweeters from mentions and replies

Beyond this, we study difference in the key players in the Twitter network over each time period. We measure importance using eigenvector centrality, and plot the centrality measure for the anonymized top five members in Fig. 5. We see that the most important tweeter, Tweeter A, is far more central than the other tweeters. While tweeter A maintains highest centrality, tweeters B-E jockey for the second highest. Other than A, each tweeter is more central in a different segment of the ongoing conversation. Also, we see that second time segment has no tweeters much higher in centrality than another. This indicates a different type of conversation structure than the other segments.

Case study conclusion

We demonstrated that the internal/external similarity approach can be used in a streaming context. While we only used two case-studies for parameter tuning, we showed that parameter tuning of this kind is possible, and yields reasonable approximations of prior results. Applying this to our Twitter data, we found 5 segments, each with different network characteristics and central tweeters. While we have shown that this type of analysis is feasible and gives interesting segments, many more case-studies must be used to optimize the parameters before this can be used generally.

Threats to validity and further work

The networks examined here were small enough to be easily stored in memory. For much larger networks this method might be too computationally expensive. For large networks, calculations with sparse matrices would have to be implemented. For many time slices, aggregation or artificial splitting of the data before analysis could be performed to reduce runtime. For a very long dataset, analyzing each half separately would cut runtime in half. Future work may entail how aggregation and splitting could be used to determine interesting regions, which could then be decomposed and analyzed in detail.

In terms of our comparison to Gernealized Louvain, we only performed a grid search over 286 parameter combinations. It is possible that a more fine-grained search would have led to better segmentation than that which we found. Our goal here was to point out

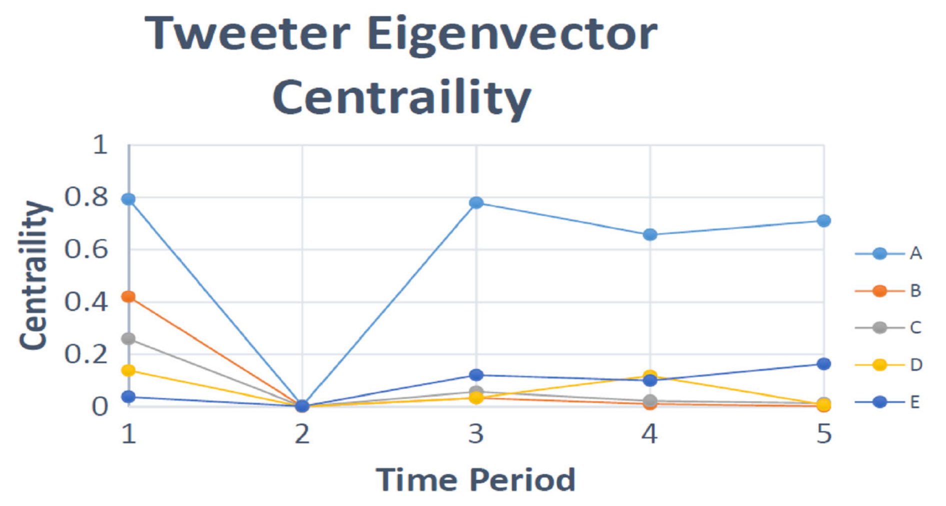

Fig. 5 Eigenvector centrality for the top 5 tweeters is shown for each network segment. Here we see Tweeter A dominates the conversation in general, while the other 4 tweeters jockey for second place. Also, the second segment's network is concentrated such that no one tweeter is much more central than other

this limitation, and the lack of ability to determine an appropriately large search space. Also, we acknowledge that the Generalized Louvain algorithm is not trying to optimize our similarity functions, so it is expected that it will not reach an optimal solution. However, since our similarity function is based on first principles of networks, we believe any temporal partitioning algorithm should performed reasonably well on our measures.

A major simplification was made in the streaming algorithm when the adjacency matrix was used in place of the the co-group matrix. Given the similarity values of 0.84 and 0.81, we believe that this simplification is worth the trade-off in computational cost. If one does not believe this, or if the computational cost is not a problem, the co-group matrix could easily still be used. However, the streaming parameters found here would probably not be applicable. More generally, the streaming parameters here were found as a proof of concept. Parameter for a streaming algorithm would have to be optimized over far more than 2 networks to be considered truly valid generally.

Conclusion

We began by introducing a simple algorithm for dynamic partitioning of networks with static nodesets, through the product-moment correlation. Applying this to the 7th convocation of Ukraine's legislature, the Verkhovna Rada, showed a large disruption during the revolution. Specifically, the majority faction split and some joined with their opposing group, leading to three factions similar in size. We compared this to the Generalized Louvain algorithm, and showed that it was impractical to get similarly optimized results due to its parameter tuning problem.

Then, we extended the algorithm to semi-static nodesets, and applied this to the Enron email dataset. We found only one partition, significantly less than found with GraphScope. We hypothesized that this is largely due to our different approach, and that streaming algorithms tend to give more partitions.

Lastly, we proposed an anomaly-detection-based streaming alternative. This algorithm's parameters were optimized over the two previous case studies. The streaming algorithm was applied to Twitter data obtained from Ukraine over 6 months. From this data four partitions were found, in which the network segment's characteristics, central tweeters, and topic was slightly different, though most revolved around the K-Pop group BTS. Effectively, we demonstrated that similarity-based anomaly detection is a feasible method of segmenting social media data in real time.

As dynamic network data becomes more and more ubiquitous, tools and frameworks to analyze such networks become more and more important. This work allows the vast variety of static community detection tools to be used in a dynamic analysis. Our methods allow researchers to test the harm of collapsing their temporal data into a single network. If harm is done, partitions minimizing this harm are given.

Acknowledgments
We would like to thank Tymofiy Mylovanov for his expert knowledge of the Rada, and for his insights into the events detected with our algorithm. We would also like to thank MAJ David Beskow, for collecting the Twitter data used in the last case study.

Funding
This work was supported in part by the Office of Naval Research (ONR) through a MURI Grant N00014-17-1-2675, and the Center for Computational Analysis of Social and Organization Systems (CASOS). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Office of Naval Research or the U.S. government. Thomas Magelinski was also supported by an ARCS Foundation scholarship.

Availability of data and materials

The Enron and Rada datasets supporting the conclusions of this article are included within the article and its additional files. The Twitter data cannot be shared due to Twitter's data handling policies.

Authors' contributions

TM and KMC planned the overall project. KMC supervised the entire project. TM performed the analysis. Both authors have read and approved the manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Received: 7 November 2018 Accepted: 24 April 2019
Published online: 28 May 2019

References

Aggarwal C, Subbian K (2014) Evolutionary network analysis: A survey. ACM Comput Surv (CSUR) 47(1):10
Aynaud T, Fleury E, Guillaume JL, Wang Q (2013) Communities in evolving networks: definitions, detection, and analysis techniques. In: Dynamics On and Of Complex Networks. Springer, New York Vol. 2. pp 159–200
Aynaud T, Guillaume JL (2010) Static community detection algorithms for evolving networks. In: Modeling and optimization in mobile, ad hoc and wireless networks (WiOpt), 2010 proceedings of the 8th international symposium on. IEEE: conference publishing services. pp 513–519
Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech: Theory Exp 2008(10):P10008
Duncan M (1958) Dimensions of Congressional Voting: A Statistical Study of the House of Representatives in the Eighty-First Congress, Vol. 1. University of California Press, Oakland
Goldberg MK, Magdon-Ismail M, Nambirajan S, Thompson J (2011) Tracking and predicting evolution of social communities. In: SocialCom/PASSAT. IEEE: conference publishing services. pp 780–783
Greene D, Doyle D, Cunningham P (2010) Tracking the evolution of communities in dynamic social networks. In: Advances in social networks analysis and mining (ASONAM), 2010 international conference on. IEEE: conference publishing services. pp 176–183
Krackhardt D (1987) Cognitive social structures. Soc Networks 9(2):109–134
Li Z, Sun D, Xu F, Li B (2017) Social network based anomaly detection of organizational behavior using temporal pattern mining. In: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. IEEE: conference publishing services. pp 1112–1119
Lin YR, Chi Y, Zhu S, Sundaram H, Tseng BL (2008) Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In: Proceedings of the 17th international conference on World Wide Web. ACM, New York. pp 685–694
Lin YR, Chi Y, Zhu S, Sundaram H, Tseng BL (2009) Analyzing communities and their evolutions in dynamic social networks. ACM Trans Knowl Disc Data (TKDD) 3(2):8
Masuda N, Holme P (2019) Detecting sequences of system states in temporal networks. Sci Rep 9(1):795
Matias C, Miele V (2017) Statistical clustering of temporal networks through a dynamic stochastic block model. J R Stat Soc Ser B Stat Methodol 79(4):1119–1141
McCulloh I, Carley KM (2011) Detecting change in longitudinal social networks. Military Academy West Point NY Network Science Center (NSC), Fort Belvoir. Tech. rep.
Miller H, Mokryn O (2018) Size agnostic change point detection framework for evolving networks 1809:09613. arXiv preprint arXiv
Morini M, Flandrin P, Fleury E, Venturini T, Jensen P (2017) Revealing evolutions in dynamical networks 1707:02114. arXiv preprint arXiv
Mucha PJ, Richardson T, Macon K, Porter MA, Onnela JP (2010) Community structure in time-dependent, multiscale, and multiplex networks. Science 328(5980):876–878
Newman ME (2004) Fast algorithm for detecting community structure in networks. Phys Rev E 69(6):066133
Ng AY, Jordan MI, Weiss Y (2002) On spectral clustering: Analysis and an algorithm. In: Advances in neural information processing systems. MIT Press, Cambridge. pp 849–856
Peel L, Clauset A (2015) Detecting change points in the large-scale structure of evolving networks. AAAI Press, Palo Alto
Peel L, Larremore DB, Clauset A (2017) The ground truth about metadata and community detection in networks. Sci Adv 3(5):e1602548
Poole KT (2005) Spatial Models of Parliamentary Voting. Cambridge University Press, Cambridge
Poole KT, Rosenthal H (1985) A spatial model for legislative roll call analysis. Am J Polit Sci 29(2):357–384
Priebe CE, Conroy JM, Marchette DJ, Park Y (2005) Scan statistics on enron graphs. Comput Math Org Theory 11(3):229–247
Rossetti G, Cazabet R (2018) Community discovery in dynamic networks: a survey. ACM Comput Surv (CSUR) 51(2):35
Rowe R, Creamer G, Hershkop S, Stolfo SJ (2007) Automated social hierarchy detection through email network analysis. In: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis. ACM, New York. pp 109–117
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. ACM, New York. pp 687–696
Taylor D, Caceres RS, Mucha PJ (2017) Super-resolution community detection for layer-aggregated multilayer networks. Phys Rev X 7(3):031056
Verkhovna Rada of Ukraine LawsofUkraine (2018). https://zakon.rada.gov.ua/laws/main/en/annot Accessed 2017 July