Automatic Segmentation of Dynamic Network Sequences with Node Labels

Sorour E. Amiri, Liangzhe Chen, and B. Aditya Prakash

Abstract—Given a sequence of snapshots of flu propagating over a population network, can we find a segmentation when the patterns of the disease spread change, possibly due to interventions? In this paper, we study the problem of segmenting graph sequences with labeled nodes. Memes on the Twitter network, diseases over a contact network, movie-cascades over a social network, etc. are all graph sequences with labeled nodes.
Most related work on this subject is on plain graphs and hence ignores the label dynamics. Others require fix parameters or feature engineering. We propose SNAPNETS, to *automatically* find segmentations of such graph sequences, with different characteristics of nodes of each label in adjacent segments. It satisfies all the desired properties (being parameter free, comprehensive and scalable) by leveraging a principled, multi-level, flexible framework which maps the problem to a path optimization problem over a weighted DAG. Also, we develop the parallel framework of SNAPNETS which speeds up its running time. Finally, we propose an extension of SNAPNETS to handle the dynamic graph structures and use it to detect anomalies (and events) in network sequences.
Extensive experiments on several diverse real datasets show that it finds cut points matching ground-truth or meaningful external signals and detects anomalies outperforming non-trivial baselines. We also show that the segmentations are easily interpretable, and that SNAPNETS scales near-linearly with the size of the input. Finally, we show how to use SNAPNETS to detect anomaly in a sequence of dynamic networks.

Index Terms—Segmentation, Graph, Sequence.

1 Introduction

Suppose we have a sequence of Ebola infections and the associated contact network of who-can-infect-whom. Can we quickly tell a public health expert when the infection patterns change possibly due to a virus mutation? By itself, it is crucial for public health to understand the virus propagation and to design a good immunization strategy. One possible approach is to segment the sequence based on some manually selected features, such as the rate of infections. However by directly analyzing the underlying social network, and using both the infected and uninfected nodes, we can improve the segmentation as well as its interpretability (e.g. 'disease spread in a tree like fashion among elderly till Monday, and changed to clique-like fashion among the young' and so on).

Segmenting a graph sequences is an important problem which can help us in better understanding the evolution of the dataset. This problem has numerous applications from epidemiology/public health to social media (rumors/memes on social networks like Twitter), anomaly detection and cyber security (malware on computer networks). In this paper, we study the problem of segmenting a graph sequence with varying node-label distributions. First, we assume binary labels and fixed structure for graphs. Next, we propose an extension to our algorithm to handle dynamic graphs with varying nodes and edges then leverage it to detect anomalies and important events. For diseases/memes, the labels can be 'infected'/'active' (1) & 'healthy'/'inactive' (0), and the network can be the underlying contact-network. Our problem is:

PROBLEM 1: SEGMENTATION

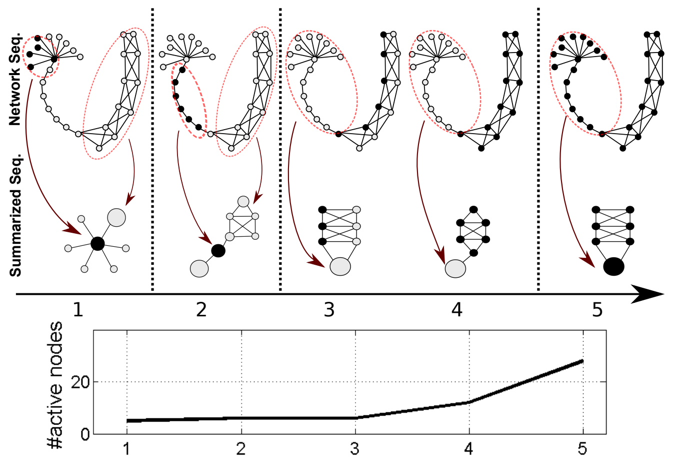


Figure 1: **TOY EXAMPLE: SNAPNETS automatically identifies four significant steps of the network sequence. The extracted time series (e.g. #active nodes) can not capture a proper segmentation. Gray nodes are inactive (i.e. label 0), and black nodes are active (i.e. label 1).**

Given: a sequence $\mathcal{G}$ of networks $G_1, G_2, \ldots, G_T$ with labeled nodes,

Find: best segmentation c^*, which captures different patterns of node labels in $\mathcal{G}$ such that adjacent segments have different characteristics of nodes with the same label.

TOY EXAMPLE. Suppose $\mathcal{G} = \{G_1, G_2, G_3, G_4, G_5\}$ with 0, 1 labeled nodes (Fig. 1 top row). There are four main steps in $\mathcal{G}$: First a central node in a star and some of its spokes have the label 1; next, low degree nodes in a chain-shaped manner get label 1 (structural change). In the third segment, the label moves to another community of the graph (community change). Finally,

• *Sorour E. Amiri, Liangzhe Chen and B. Aditya Prakash are with the Department of Computer Science, Virginia Tech.*
E-mail: {esorour, liangzhe, badityap}@cs.vt.edu

Properties	SNAPNETS	Feature eng. and time series [1], [2], [3]	Plain-graph-based [4], [5], [6], [7]
Parameter free	✓	✗	✗
Comprehensive	✓	✓	✗
Scalable	✓	✗	✓

Table 1: **Comparison of SNAPNETS with alternative approaches. A dashed cross means most approaches does not satisfy the property; similarly for the dashed check.**

the whole graph gets label 1 which indicates an activation rate increase in the network (rate changes). Hence $c^* = \{1, 2, 3, 5\}$. Note that even though the 'active' sub-graphs in time-step 2 and 3 are both chains, their roles in the entire graph are different and so they should belong in different segments. In time-step 2 the active chain is a bridge between two parts while the chain in time-step 3 is part of a near-clique community (role change). Therefore, to get c^* we must consider the entire graph at each time stamp.

Based on the above example, any ideal segmentation algorithm should have the following desired properties:

P1. Parameter free: Find the best number of segments and segmentation without use of parameters i.e. change threshold, time window, and number of desired segments.

P2. Comprehensive: Use the entire snapshot for segmentation, instead of merely active subgraphs.

P3. Scalable: The method must be scalable (i.e. scales sub-quadratically with the input size which can be millions of edges and nodes in the sequence).

This problem has been barely (if at all) studied in literature. Unlike most methods that we can adopt to solve this problem, SNAPNETS has all the above mentioned properties. (Table 1 shows a brief comparison; more discussion in Section 3.1). SNAPNETS is a novel multi-level approach which summarizes the given networks/labels in a very *general way* at *multiple different time-granularities*, and then converts the problem into an appropriate *optimization problem* on a data structure. We give a novel efficient algorithm for the optimization problem as well. A strong advantage of this framework is that it allows us to *automatically* find the right number of segments avoiding over or under segmentation in a very systematic and intuitive fashion. Further it gives naturally interpretable segments, enhancing its applicability. Also, we easily extend SNAPNETS to segment dynamic graph sequences which can be used to detect anomalies and important events. Finally, we demonstrate SNAPNETS's usefulness via multiple experiments for segmentation and anomaly detection on diverse real-datasets.

The rest of the paper is organized as follows: We first go over useful preliminaries in Section. 2, then give an overview of our approach in Section 3. Then we describe SNAPNETS in detail in Section 4 and propose its extension to handle dynamic graph sequence and it application to detect anomalies in Section 5 followed by empirical results in Section 6. Finally, we discuss related work and conclude in Sections 7 and 8.

2 PRELIMINARIES

In this section, we go over useful preliminaries, in particular diffusion models and plain graph summarization algorithms. Table 2 summarizes the notations that we use throughout this paper to describe the segmentation problem and SNAPNETS algorithm.

Table 2: Some of the notations used

Symbol	Description
$G_{\cdot}$, $G^c_{\cdot}$	The original and the summarized graph.
$\mathcal{G}$	A sequence of T *Act-snapshots* (i.e. *AS-Sequence*)
T	The last time stamp in *AS-Sequence* $\mathcal{G}$
c	A valid segmentation of time interval $[1, T]$
c^*	The best segmentation of time interval $[1, T]$
$\mathcal{C}$	The set of all possible segmentations.
$s_{i,j}$	A segment $[i, j)$.
$\mathcal{S}$	The set of all possible segments.
$\mathbf{F}_{\cdot}$	The feature vector of each *C-graph*
$\hat{\mathbf{F}}_{\cdot}$	The feature vector of each segment.
$d(\cdot, \cdot)$	Distance between two segments.
G_s	Segmentation graph
s , t	Source and target nodes of the segmentation graph
λ_G	The leading eigenvalue of graph G
$l_{x^{\cdot}}$	The label of a node x
ρ	The compression ratio to summarize graph G
nop	Number of processors

2.1 Diffusion models

Node labels (i.e. active:1 and inactive:0) usually come from diffusion process in networks. Different models are distinguished based on how they define the activation cycle on a node in a network G. We categorize diffusion models into two main groups: (1) Progressive: such as **I**ndependent **C**ascade (IC) and **S**usceptible-**I**nfected (SI) models where nodes can not get inactive. (2) Non-progressive: such as the 'flu-like' **S**usceptible-**I**nfected-**S**usceptible (SIS), and **S**usceptible-**I**nfected-**R**ecovered (SIR) models where active node can get inactive again. [8], [9].

In diffusion models, initially, a node is susceptible (inactive). Overtime, that node can get infected (active) according to some probability. Finally, depending on the diffusion model, the node may remain infected or become susceptible (inactive) again or be removed (immunized). For example, in the IC model a vertex $v \in V$ is called active if it has been influenced and inactive otherwise. Each active node has a single chance to activate each currently inactive neighbor with an independent probability over the edge. This diffusion process continues until no more activations are possible.

2.2 Summarizing plain graphs

We leverage graph summarization in our proposed algorithm. Our summarization is inspired by *CoarseNet* [10] method, an algorithm for coarsening a graph while preserving the propagation characteristics of the graph as much as possible. The algorithm takes a weighted graph $G(V, E, w)$ and a target fraction $0 < \alpha < 1$ as input, and coarsens a fraction $1 - \alpha$ of total nodes such that the coarsened graph $G' = (V', E', w')$ approximates graph G with respect to its diffusive properties.

CoarseNet uses the leading eigenvalue of the adjacency matrix as an indicator of the graph's diffusion properties, and greedily merges adjacent nodes such that the drop in the first eigenvalue is minimized. It evaluates the quality of merging each edge based on a score they define as follows,

$$score(a, b) = |\lambda_{G-(a,b)} - \lambda_G| = \Delta\lambda_{(a,b)} \tag{1}$$

Where, λ in the leading eigenvalue of the adjacency matrix of a graph. $score(a, b)$ measure the change of leading eigenvalue after merging a and b. It is shown that the estimation of $score(a, b)$ is as follows,

$$score(a, b) \simeq \frac{-\lambda(u_a v_a + u_b v_b) + v_a \overrightarrow{u}^T \overrightarrow{c^{o}} + \beta_2 u_a v_b + \beta_1 u_b v_a}{\overrightarrow{v}^T \overrightarrow{u} - (u_a v_a + u_b v_b)} \quad (2)$$

Where $\overrightarrow{u}$ and $\overrightarrow{v}$ are the right and left eigenvectors of graph G. u_a denotes the component of $\overrightarrow{u}$ corresponding to vertex a and $\overrightarrow{c^{o}}$ is the vector of outgoing edges from the supernode c (i.e. the node is generated after merging a and b).

3 OVERVIEW AND MAIN IDEAS

For sake of simplicity, we focus on the case when nodes have binary labels[1] i.e. active: 1 or inactive: 0. Also, for ease of description, first we assume that the network remains constant through time and we treat the problem as one with a series of graph snapshots. Next, we show a natural extension of SNAPNETS to work with dynamic graph sequences (See sections 5 and 6.8).

Finally, we allow that nodes can even *switch* between labels freely (c.f. Figure 1). This means that we can handle both progressive/non-progressive scenarios: e.g. SI, IC, and SIS models. Next we give some useful definitions:

Definition 1 (Act-snapshot). $G(V, E, \mathbf{L})$ is an *Act-snapshot*. $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$ are sets of nodes and edges of G. $\mathbf{L} = [l_1, l_2, \ldots, l_n]$ shows the labels of nodes. l_x is 1 if v_x is active and 0 otherwise.

Definition 2 (AS-Sequence). $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$ is sequence of T *Act-snapshots* with G_i at time-step i.

Definition 3 (Segment). A segment $s_{i,j}$ is a time interval between *Act-snapshots* G_i and G_j i.e. $s_{i,j} = \{[i, j) \mid i < j\}$. Set of all possible segments is $\mathcal{S} = \{s_{1,2}, s_{1,3}, \ldots\}$.

Definition 4 (Segmentation). A segmentation c of size q is a partition of time interval $[1, T]$ with q time stamps i.e. $c = \{a_1, a_2, \ldots, a_q\}$ where $a_i \in \{1, 2, \ldots, T\}$. The set of all possible segmentations is $\mathcal{C}$.

Definition 5 (Act-set$_i$). *Act-set$_i$* contains the active nodes in *Act-snapshot* G_i i.e. *Act-set$_i$* $= \{v_x | l_x = 1\}$.

Hence our problem is to automatically segment a given *AS-Sequence*. We next explain the shortcomings of alternative approaches, and then give the big picture of our framework.

3.1 Shortcomings of alternative approaches

Two natural ways to adapt existing algorithms for this task are: (a) extract complex features from *Act-snapshots* and use time-series segmentation; and (b) extract *Act-set*s and use plain-graph-based methods.

Feature Eng. and Time Series. Converting graphs sequences to time series has several drawbacks. First of all, it needs laborious feature-engineering: designing the right features to capture the pattern of graphs is a complicated task, usually requires a large set of expensive features and the best choice of features may differ for different sequences [11]. For example, we may need to extract the number of stars in a sequence of social networks to identify the structure of graphs. However, we may need to count the number of ladders in infrastructure networks. Second, typical time series segmentation algorithms, which use "local" change detection, do not satisfy our desired properties. They usually need a threshold [2] (which usually depends on knowing the number of desired segments) to detect a change; or they fix *one* aggregation time period for the tracking [1]. All of these can be problematic, as it is fundamentally hard to set these parameters.

Plain-graph-based analysis. Instead of manually designing complicated features, an alternative is to use plain-graph-based methods on induced subgraphs from *Act-snapshots*. However, these approaches do not satisfy **P2**, as they typically track only the *Act-set* in each *Act-snapshot* [4], [5], [7]. As we show in Figure 1, using only the active sub-graphs leads to less meaningful segmentation: the active sub-graphs at time-step 2 and 3 are both a chain of a same size, nevertheless, as discussed before the roles of these chains are different in the two snapshots. If we just track active sub-graphs, we cannot detect this difference.

3.2 Overview of our method SnapNETS

In order to overcome the disadvantages we discussed before, we propose a "global" framework which looks at the entire *AS-Sequence* $\mathcal{G}$ and computes the correct segmentation. Due to **P1**, we want to examine all possible segmentations $\mathcal{C}$ over all granularities. How to do this efficiently? Our first main idea is to use a graph data structure (called the *segmentation graph* $\mathsf{G_s}$) to efficiently represent the exponential number of all segmentations in space polynomial with respect to the sequence length. The nodes mainly represent the segments in $\mathcal{S}$, while the edge weights indicate the distance ('difference') between adjacent segments. Hence any segmentation is mapped to a path between start and end time in $\mathsf{G_s}$.

How to compute the distance between any adjacent segments $w(s_{i,j}, s_{j,k})$ (each segment will contain sets of *Act-snapshots* G_i)? Note that we want to use the entire graph due to **P2**. This is related to the problem of graph isomorphism. It is natural to compare the segments by examining extracted features such as number of nodes, cliques, diameter and so forth. This will require complex feature construction to correctly and sufficiently represent each graph snapshot. Nevertheless, despite the size of the graphs, patterns in the real-world are usually much less complex. Hence, our second main idea is to develop a *smaller* summary G_i^c which maintains important information in an efficient manner. As a result, we only need a few standard features to represent these summaries.

Finally, how to find the best path in $\mathsf{G_s}$? We need to define this best path and design an efficient algorithm to find it in $\mathsf{G_s}$. Our third main idea is to use the average longest path optimization problem on $\mathsf{G_s}$, as it intuitively regularizes the length of the path (number of segments) with the weight (difference between segments). We also develop the efficient novel algorithm LAYERED-ALP to find this path.

In short, we pursue 3 main goals:

Goal 1. *Summarizing G_i*: summarize each G_i to capture the structural and label dependent properties (Section 4.1).

Goal 2. *Constructing* $\mathsf{G_s}$: extract the features of summarized graphs and compute the distance between adjacent segments; and then construct $\mathsf{G_s}$ (Section 4.2).

Goal 3. *Defining and extracting the best segmentation*: define the best segmentation (path) and develop an efficient algorithm to find it (Section 4.3).

1. Extending to multiple labels is interesting future work.

Figure 2 shows an overview of SNAPNETS (**Snap**ping **NET**work **S**equences). Our careful design and parallel implementation help to satisfy **P3**: SNAPNETS is sub-quadratic in running time with respect to the size of the sequence and our experiments show its effectiveness and scalability.

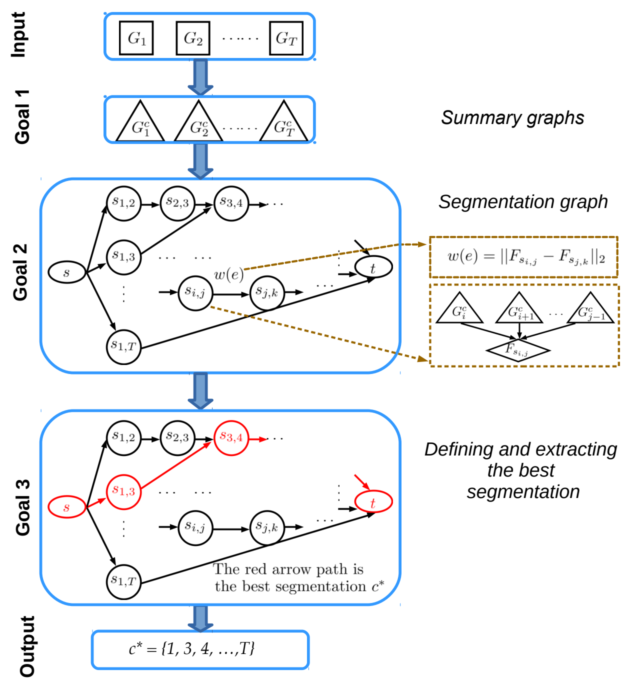


Figure 2: **SNAPNETS overview (Best viewed in color).**

4 SNAPNETS: DETAILS

4.1 Goal 1: Summarizing Act-snapshots

We first propose finding a *C-graph* (i.e. G_i^c), which summarizes the structural properties and the nodes labels of each *Act-snapshot* G_i. Popular methods for graph summarization include graph sparsification [12] which try to carefully remove edges to reduce the graph's density while maintaining some properties. Nevertheless, these methods are typically designed for plain graphs and it is not straightforward to modify them for *Act-snapshots*. So we adopt a different, *merging-based* approach which reduces the *number of nodes* instead while maintaining an intuitive and important property.

Role of Eigenvalues: In many real datasets node labels come from a diffusion/propagation process. Recent work [13] shows that important diffusion characteristics of a graph (including the so-called 'epidemic threshold') are captured by the leading eigenvalue of the adjacency matrix, for *almost all* cascade models. This naturally suggests that if the leading eigenvalue of the adjacency matrix of the summarized graph G_i^c and *Act-snapshot* are close, G_i and G_i^c will have similar properties.

Summarizing Act-snapshots via Coarsening: Motivated by the above, we want to successively merge connected nodes into 'super-nodes' (i.e. 'coarsen') while maintaining the leading eigenvalue of the adjacency matrix. Also, we want to keep the same set of labels (0/1) in the *C-graph* to keep it consistent with the *Act-snapshot* and help interpretability and make it easier to compare *C-graphs*. Thus, we define the summarization problem as follows,

PROBLEM 2: *Act-snapshot* SUMMARIZATION

Algorithm 1: *Act-snapshot* summarization

```
Data: Act-snapshot $(V, E, \mathbf{L})$, $\rho$
Result: summary graph C-graph
1 for $(x, y) \in E$ do
2     Compute $score(x, y)$;
3 $\pi \leftarrow$ Sort the edges base of their scores in increasing order;
4 $r = 0$, $G_:^c = G$;
5 while $r \leq (1 - \rho) \cdot |V|$ do
6     $(x, y) = \pi(r)$;
7     if $l_x == l_y$ then
8         $G_:^c \leftarrow merge_{G_:^c}(x, y)$;
9         $r \leftarrow r + 1$;
```

***Given*:** an *Act-snapshot* $G_i(V_i, E_i, L_i)$, and remained fraction of nodes ρ.

***Find*:** a coarsened graph $G_i^c(V_i^c, E_i^c, L_i^c)$ such that

$$\underset{|V_i^c| = \rho |V_i|}{\textit{minimizes}} \ |\lambda_{G_i} - \lambda_{G_i^c}| \quad \textit{subject to} \ \sum_{(x,y) \in E_i \text{ is merged}} |l_x - l_y| = 0$$

Here λ_G is the leading eigenvalue of graph G and l_x is the label of node x. This formulation allows us to be model-free and not assume any specific model (such as IC/SIS, etc.). The constraint in PROBLEM 2 maintains the 'frontier' between active and inactive nodes to help consistency and interpretability. PROBLEM 2 is similar to the graph coarsening problem (GCP [10]) whose only goal is to maintain λ_G, but without any constraint—they give an efficient algorithm for this purpose which merges edges based on a quality score. Hence, we modify that algorithm by prohibiting merging of node-pairs with different labels. Algorithm 1 shows our algorithm to solve the PROBLEM 2. We compute the scores of edges based on Equation 2 and sort them in lines 1-3. Then, gradually merge edges with end nodes with same labels from the ordered list (lines 5-9). This algorithm works very well in practice and gives near-linear running time. Note that a better algorithm for PROBLEM 2 will only improve our results. A desired value for ρ is the lowest value which maintain λ_G. We use the same amount of coarsening ($\rho = 0.1$) as in [10]. If the ρ value is too large, the summary will be basically similar to the original graph, and we need more complicated features to identify its properties (similar to the original graphs). Also, too small ρ value (i.e., lower than 0.1) is not desirable since all the nodes collapse into few super-nodes and the λ will deteriorate fast.

Figure 1 shows our summaries via PROBLEM 2 for the TOY EXAMPLE: the *C-graphs* clearly show the important non-trivial pattern changes in both the structural and label properties of the original graphs succinctly.

4.2 Goal 2: Constructing the segmentation graph

After summarizing the *Act-snapshots*, each segment in the *AS-Sequence* contains a set of *C-graphs*. How to find the distance between two such segments? In general, computing distances between *unlabeled* graphs is itself a challenging problem [5]. Fortunately, in our case, we can just extract *simple* features from the *C-graphs* due to their small size and complexity; and use them to compute the distance. Subsequently, we build the segmentation graph $\mathsf{G_s}$ to store the segments and distances information. Recall that $\mathsf{G_s}$ can efficiently represent all the exp. number of possible segmentations in poly. space.

Feature extraction of C-graphs Extracting features from G_i^c is

Type	ID	Name	Features description
Structural	f_1	Largest eigenvalue of the adjacency matrix	It indicates the structural and label dependent properties of *C-graphs*. We expect changes in eigenvalues of *C-graphs* in *AS-Sequence* due to the restriction of maintaining the frontier (Sec 4.1). Hence this feature will encode how the labels are distributed with respect to the *Act-snapshot*.
	f_2	Number of edges	It measures the density of each *C-graph* which indicates how nodes got merged due to the label distribution in the *Act-snapshot*
	f_3	Entropy of the edge weight distribution	It encodes which edges in the original graph got merged. During *CoarseAct*, the edge weights change to maintain the leading eigenvalue (see Sec 4.1). Therefore, the distribution of edge weights contains information about which edges are merged and how much merging happens.
	f_4	Average clustering coefficient	It measures the relative frequency of triangles in a *C-graph*. It shows when a new community get activated.
Label based	f_5	Number of active nodes	It counts the remaining active nodes after summarizing an *Act-snapshot*.
	f_6	Average PageRank of active nodes	It measures the structural importance of active nodes in *C-graphs*.
	f_7	Average degree of active nodes	It measures centrality among active nodes in *C-graphs*.
	f_8	Average degree of active neighbors of active nodes	It indicates the role and importance of active nodes.

Table 3: **Features extracted to represent each summarized *Act-snapshot* (i.e. *C-graph*).**

much more efficient primarily because of their smaller size. Further our summarization maintains the relevant important properties effectively. So we do not need complex features such as “number of particular substructures” (e.g. stars, maximal cliques, ladders, etc.) used in related work.

We extracted multiple *standard* features [14] and eliminate correlated ones to get eight features for each G_i^c (See Table 3 for a description). Feature vector $\mathbf{F}_i$ contains: *Structural* features (f_1-f_4); and *Label dependent* features (f_5-f_8) (label-dependent properties). Finally, we normalize them by range normalization for a meaningful comparison between the features [14]. Thanks to our careful design, we can use very simple features for our task.

Segmentation graph We now describe how to construct $\mathsf{G_s}$ to compactly store and represent segmentations. $\mathsf{G_s}(V_s, E_s)$ is a unique weighted DAG where:

Nodes (V_s): For each segment $s_{i,j} \in \mathcal{S}$, there is one node in the graph $\mathsf{G_s}$. We also add two extra nodes to the graph: a source node s and a target node t. Therefore, $V_s = \mathcal{S} \cup \{s, t\}$.

Edges (E_s): There is a directed edge from node $s_{i,j}$ to any node $s_{j,k}$. Also, the source node s links to all nodes with starting time stamp 1 and all nodes with ending time stamp T links to the target node t. Hence, $E_s = \{e(s_{i,j}, s_{j,k})\} \cup \{e(s, s_{i,j}) | i = 1\} \cup \{e(s_{i,j}, t) | j = T\}$.

Edge Weights ($w(e)$): The weight of all edges from s or to t are zero. The weight of an edge from $s_{i,j}$ to $s_{j,k}$ is equal to the distance between sets of *C-graphs* in their corresponding segments i.e. $w(e(s_{i,j}, s_{j,k})) = d(s_{i,j}, s_{j,k})$.

How to get this distance? Using the $\mathbf{F}_i$ for each G_i^c, we compute the average feature vector over all the *C-graphs* in a segment as the segment's representative i.e. $\widehat{\mathbf{F}}_{s_{i,j}} = \frac{\sum_{a=i}^{j} \mathbf{F}_a}{(j-i+1)}$, where $\mathbf{F}_a$ is the feature vector of G_a^c in $s_{i,j}$. This representation has a natural interpretation as it captures the average ‘pattern’ of *C-graphs* of the segment. Then the distance $d(s_{i,j}, s_{j,k})$ between ‘$s_{i,j}$’ and ‘$s_{j,k}$’ can be defined as $d(s_{i,j}, s_{j,k}) = ||\widehat{\mathbf{F}}_{s_{i,j}} - \widehat{\mathbf{F}}_{s_{j,k}}||_2$.

Figure 3 shows the $\mathsf{G_s}$ for our Toy Example. Edge weights are not shown for clarity. Note that $\mathsf{G_s}$ is a DAG since its edges are directed and there is no cycle in it (as we cannot go back in time). Further, note that we can compute the distance for every edge in $\mathsf{G_s}$ independently and in parallel. Also, we need to compute the summary just once for each G_i, *not* for each segment in $\mathsf{G_s}$.

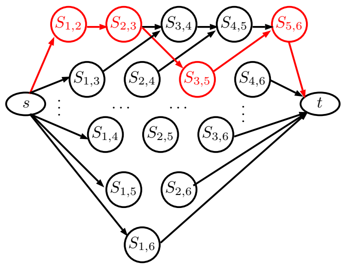


Figure 3: **The segmentation graph $\mathsf{G_s}$ of the Toy Example. The red path from s to t represent the best segmentation c^*.**

4.3 Goal 3: Finding the best segmentation

Let $\mathcal{P}$ be the set of all paths in $\mathsf{G_s}$ from s to t. Then,

Lemma 1. Each path $p \in \mathcal{P}$ corresponds to a valid segmentation $c \in \mathcal{C}$ and for each $c \in \mathcal{C}$ there is a path $p \in \mathcal{P}$.

Proof 1. It is obvious based on construction of $\mathsf{G_s}$.

Hence to get the best segmentation, we only need to *define* and *find* the best path in $\mathsf{G_s}$; which we discuss next.

Average longest path Note that defining the best path is a different and independent question to that of defining edge weights. We define the best segmentation as follows:

Problem 3: Finding the best segmentation

***Given*:** a segmentation graph $\mathsf{G_s}$

***Find*:** the average longest path from s to t in $\mathsf{G_s}$ i.e.

$$c^* = \arg\max_{c \in \mathcal{P}} \frac{\sum_{s_{i,j}, s_{j,k} \in \mathcal{S}} w(e(s_{i,j}, s_{j,k}))}{|c|} \quad (3)$$

Thus, Problem 3 is the *Average*-Longest Path (ALP) problem on $\mathsf{G_s}$ (restricted to the path set $\mathcal{P}$). ALP defines the path (segmentation) quality as the average value of edge weights in the path (distance between its segments). Note that ALP is parameter-free and importantly, it also naturally *balances* the ‘length’ (weight) of the path (difference between segments) with number of nodes in the path (number of segments).

An alternative ‘parameter-free’ optimization would have been the Longest Path (LP) problem: which will try to find the *longest* (heaviest) path in $\mathcal{P}$ (Equation 3, without the denominator). However, the LP formulation will suffer from *over-segmentation*—it is biased by the number of segments in the path, in the sense that it tends to prefer longer paths with more nodes, irrespective of the edge weights [15]. In practice our observations confirm that LP contains unnecessary edges with low weight (see Sec 6). Our ALP objective is intuitive and overcomes the disadvantage of LP. Figure 3 shows the ALP for TOY EXAMPLE in red.

4.3.1 LAYERED-ALP

ALP can be solved in polynomial time on DAGs (recall $\mathsf{G_s}$ is a DAG)[2]. Current state-of-the-art algorithm [15] can solve PROBLEM 3 in $O(V_s^2 \cdot E_s)$. This is too slow for our purposes; hence, we propose a new and more efficient $O(E_s)$ algorithm for ALP on DAGs called LAYERED-ALP.

The main observation we use is that the ALP from s to t is the *longest* ('heaviest') path among all paths with the same number of nodes (the 'length') as the ALP. This fact leads us to calculate all the heaviest paths with different lengths in $\mathcal{P}$ and find the one which gives the maximum average edge weight. In LAYERED-ALP (Algorithm 3) we build a queue of layers of $\mathsf{G_s}$ (lines 5 - 7). Each layer i contains nodes which can reach t by i steps. When we iterate through layers (lines 10 - 15), we maintain the weight ($P_i(v,t)$) of the heaviest path from v to t in i steps, as well as the next node of v in this path (κ_v^i). If s is in layer i, it means there is a path from s to t. We store the $P_{CL}(s,t)$ as the *longest* ('heaviest') path with i nodes from s to t. After iterating over all layers, we have a set of *longest* paths with various number of nodes. The ALP is a path among this set which maximizes its weight divided by its number of nodes. Algorithm 3 shows the pseudo-code of LAYERED-ALP.

Lemma 2. The LAYERED-ALP algorithm correctly finds the average longest path $\mathsf{G_s}$.

Proof 2. Assume $P_i(a,b)$ is a path from node a to b with i steps. *First* we show that LAYERED-ALP finds all longest paths with different lengths from $\mathcal{P}$. The longest path problem in $\mathsf{G_s}$ has optimal sub-structures because it is a DAG: If we decompose the path $P_h(s,t)$ into $P_{h-h'}(s,v)$ and $P_{h'}(v,t)$ where v is in layer h' in the Queue, and if $P_h(s,t)$ is the longest path form s to t with h steps, then $P_{h'}(v,t)$ will be the longest path from v to t with h' steps. *Second*, ALP is a path in the set of longest paths of different lengths. Assume not, which means there is a path with the same number of nodes as one of the LPs and higher weight which is a contradiction. □

Lemma 3. Time complexity of LAYERED-ALP is $O(|E_s|)$, where $|E_s|$ is number of edges in $\mathsf{G_s}$.

Proof 3. The time complexity of LAYERED-ALP is equal to the number of edges visited in the algorithm which can be calculated as follows:

$$
\begin{array}{ll}
\text{Layer 1: } (T-1) & = (T-1) \\
\text{Layer 2: } (T-2)+(T-3)+\ldots+1 & = \dfrac{(T-1)(T-2)}{2} \\
\text{Layer 3: } (T-3)+(T-4)+\ldots+1 & = \dfrac{(T-2)(T-3)}{2} \\
\text{Layer 4: } (T-4)+\ldots+1 & = \dfrac{(T-3)(T-4)}{4} \\
\vdots & \vdots \\
\text{Layer } T\text{-1: } 1 & = 1
\end{array}
$$

Therefore, the number of visited edges is,

$$
(T-1) + \underbrace{\frac{(T-1)(T-2)}{2} + \frac{(T-2)(T-3)}{2}}_{(T-2)^2} + \underbrace{\frac{(T-3)(T-4)}{2} + \frac{(T-4)(T-5)}{2}}_{(T-4)^2} + \ldots + 1
$$

$$
= (T-1) + \sum_{i=1}^{\frac{T}{2}} (T-2i)^2 = O(T^3) = O(|E|). \quad \square
$$

2. ALP is NP-hard on general graphs

4.4 Parallelization

Summarizing different *Act-snapshots* is an entirely independent process for each *Act-snapshot* (Algorithm 2, line 1). Also, we can extract the segments features and compute the edge weights of the segmentation graph $\mathsf{G_s}$ independently (Algorithm 2 lines 2-3). These observations motivates us to propose a parallel framework for SNAPNETS to further scale it up. We divide this frameworks into two steps: (1) Summarizing *Act-snapshots*, and extracting features of each *C-graph* (2) Constructing the segmentation graph—Constructing nodes, and computing edge weights. In the following we explain each step in detail.

4.4.1 Get C-graphs and their features

If the *Act-snapshots* in the sequence are large, even the near-linear time complexity of summarization (Algorithm 1) will be expensive and time consuming. Therefore, we want to parallelize this step in Section 4.1 to further scale up SNAPNETS. The summarization process and extracting features of summary graphs are independent for each *Act-snapshot*. Therefore, we consider the following parallelization scheme to get *C-graphs* and their feature vectors: Assume there are nop processors available. (1) Assign $\frac{T}{nop}$ snapshots of the *AS-Sequence* to each processor. (2) In each processor, summarize *Act-snapshots* into *C-graphs*. (3) In each processor, extract the features of *C-graphs*. Note that the parallelization scheme needs no synchronization step which makes it extremely efficient.

4.4.2 Generate the segmentation graph $\mathsf{G_s}$

If the *AS-Sequence* is long and has many *Act-snapshots*, merely using the parallelization suggested in Section 4.4.1 will not be enough to get the best segmentation fast. Thus, we propose a method to generate the segmentation graph $\mathsf{G_s}$ in parallel. We consider the following parallelization scheme to generate $\mathsf{G_s}$: Assume that *C-graphs* features are global information, there are nop processors available and $\mathcal{S}$ is the set of all possible segments. (1) Assign $\frac{|\mathcal{S}|}{nop}$ segments to each processor. (2) Next, compute the feature representative $\hat{\mathbf{F}}$ of each segment in each processor. The $\hat{\mathbf{F}}$ of a segment $s_{i,j}$ is the average vector of all the *C-graphs* features that are in the segment $s_{i,j}$ (4) Synchronize the results and compute the edge weights of the segmentation graph $\mathsf{G_s}$.

Algorithm 2: SNAPNETS

```
Data: AS-Sequence $\mathcal{G}$
Result: The optimal segmentation $c^*$
1 For each $G_i \in \mathcal{G}$ coarsen and get $G_i^c$ once (Section 4.1);
2 Compute feature $\widehat{\mathbf{F}}$ vector of segments in $\mathcal{S}$ (Section 4.2);
3 Generate the segmentation graph $\mathsf{G_s}$ (Section 4.2);
4 $c^*$ = LAYERED-ALP ($\mathsf{G_s}$, $\mathcal{G}$, $s$, $t$ ) (Section 4.3);
```

4.5 The complete algorithm

In summary SNAPNETS has four main steps:
(1) Summarize *Act-snapshots* in *AS-Sequence*.
(2) Extract features from summarized graphs (*C-graphs*).
(3) Construct the segmentation graph $\mathsf{G_s}$ and compute the distance $d(s_{i,j}, s_{j,k})$ between any adjacent segment based on their representatives ($\widehat{\mathbf{F}}_{s_{i,j}}$ and $\widehat{\mathbf{F}}_{s_{j,k}}$) in a parallel manner.
(4) Compute the best segmentation by finding the ALP.

Algorithm 2 shows the final pseudo code of SNAPNETS.

Lemma 4. SNAPNETS has sub-quadratic time complexity $O(E \cdot \log E + \frac{E}{nop} + E_s)$, where E is the total number of edges in $\mathcal{G}$ and nop is number of processors.

Proof 4. In step 1 and 2, feature extraction is very fast due to small size of the *C-graphs* compared with *Act-snapshots*. Hence the computational cost comes from coarsening which is $O(E \log E + \frac{E}{nop})$: $O(E \log E)$ to sort edges [10] and $O(\frac{E}{nop})$ for parallel coarsening. Step 3 and step 4 take $O(E_s)$. Generating the segmentation graph $\mathsf{G_s}$ is $O(\frac{E_s+V_s}{nop})$ and finding the average longest path is $O(E_s)$ which is $O(T^3)$. In most large real world datasets (e.g. see [4]), the size of each graph (orders of tens of thousands) is usually much higher than the length of the sequence (order of 100s). Therefore, in practice, $O(E_s)$ is not the bottleneck of SNAPNETS and we can assume the last two steps take $O(E)$. Total time complexity is $O(E \cdot \log E + \frac{E}{nop} + E_s)$. Note that it does not change even if the size and structure of *Act-snapshots* change in the *AS-Sequence*. □

Even though, the time complexity of step 3 and 4 step is $O(T^3)$ due to generating the E_s and LAYERED-ALP, the computation is fast in practice (i.e., in order of seconds). Hence, it is not the bottleneck of the SNAPNETS and taking the algorithm as a whole SNAPNETS is scalable. The reason is in real-world graph sequences $T << E$ (i.e. $T \sim O(100)$ while $E \sim O(1e+6)$). Hence, summarizing real-world graphs takes minutes or hours to complete while generating the $\mathsf{G_s}$ and LAYERED-ALP take only a few seconds.

5 EXTENDING TO DYNAMIC GRAPHS

SNAPNETS gives *Cut-points* which capture the change of pattern in the *AS-Sequence* $\mathcal{G}$. Even though we assumed the structure of *Act-snapshots* in the *AS-Sequence* is fixed, we propose a procedure to extend SNAPNETS to handle dynamic structures as well. In the following we explain this procedure in detail.

5.1 Technical details

If the structures of *Act-snapshots* are dynamic (i.e. *Act-snapshots* have various number of nodes and edges), summarizing them with the same reduction factor gives *C-graphs* with different sizes.

Algorithm 3: LAYERED-ALP

```
Data: $\mathsf{G_s}$, $\mathcal{G}$, $s$, $t$
Result: $P_{alp}$
1 Initialize Queue;
2 $Layer_0 = \{t\}$ and Queue.push($Layer_0$);
3 $Layer_1 = \{s_{i,j} | j = T\}$;
4 Queue.push($Layer_1$);
5 for k = 2 to $T$ do
6     $Layer_k = \{s_{i,j} | T - j + 1 \geq k\} \cup \{s\}$
7     Queue.push($Layer_k$);
8 $Layer_{T+1} = s$ and Queue.push($Layer_{T+1}$);
9 LL = Queue.pop(), $\kappa_t^0 = \varnothing$;
10 while Queue is not empty do
11     CL = Queue.pop() for $s_{i,j} \in CL$ do
12         $\kappa_{s_{i,j}}^{CL} = \arg\max_{s_{j,k}} P_{LL}(s_{j,k}, t) + w(e(s_{i,j}, s_{j,k}))$;
13         $P_{CL}(s_{i,j}, t) = P_{LL}(\kappa_{s_{i,j}}^{CL}, t) + w(e(s_{i,j}, \kappa_{s_{i,j}}^{CL}))$;
14     If $s$ is in CL then $lp_{CL} = P_{CL}(s, t)$;
15     LL = CL;
16 $ALP = \arg\max(\frac{lp_1}{1}, \ldots, \frac{lp_T}{T})$;
17 Extract the $P_{alp}$ using $\kappa_s^T$ to $\kappa_t^0$;
```

Therefore, the feature values (table 3) of *C-graphs* may be biased by the size of *Act-snapshots*. Thus, we can not compare *C-graphs* and compute distance between segments by simply comparing their feature vectors. We want to find an efficient way to make the *C-graphs* be in a same size and their feature vectors comparable. Towards this goal, we need different compression ratios ρ for each *Act-snapshot*: We summarize smaller graphs with smaller reduction factor and larger ones with larger ρ. Algorithm 4 shows how to compute the compression ratio of each *Act-snapshot*. First, we find the minimum and maximum number of nodes in the $\mathcal{G}$. The largest *Act-snapshot* in the $\mathcal{G}$ is coarsened the most to have the same size as the other *Act-snapshots*. The 'if' statement in line 2 of the algorithm, makes sure that the size of all *C-graphs* will be the same. Finally, in lines 6 to 7 we set the ρ_i values to get the number of desired nodes. More complicated methods to compute the ρ_i of the dynamic graph snapshots may improve our results but our approach is a simple and straightforward way to achieve this goal, and the experiments show it works properly.

Lemma 5. Time complexity of Algorithm 4 to set the ρ values is $O(T)$.

Proof 5. Finding the minimum and maximum number of nodes in the $\mathcal{G}$ (line 1) takes $O(T)$. Also, the if statement (lines 2-5) take $O(1)$. Finally, the 'for' loop runs in $O(T)$. Therefore, in overall the time complexity of Algorithm 4 is $O(T)$. □

In summary, to extend SNAPNETS and handle dynamic $\mathcal{G}$, we compute the ρ values (Algorithm 4) in the first step of SNAPNETS and the rest will be the same as in Algorithm 2.

5.2 Sample application: Anomaly and event detection

An example application of the segmentation problem is to solve the anomaly (and event) detection problem which is defined as follows,

Given a sequence of graphs $\{G_1, G_2, \ldots, G_T\}$
Find a list R of time points in $1 \leq i \leq T$, as anomalies. In the above problem, nodes do not have labels and there is no restriction

Algorithm 4: SET-ρ

```
  Data: $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$
  Result: $\rho_1, \rho_2, \ldots, \rho_T$
1 $V_{max} = \max_{i=1,\ldots,T} |V_i|, \quad V_{min} = \min_{i=1,\ldots,T} |V_i|$;
2 if $0.1 \cdot V_{max} \leq V_{min}$ then
3     $V_{goal} \leftarrow 0.1 \cdot V_{max}$;
4 else
5     $V_{goal} \leftarrow V_{min}$;
6 for $i = 1, 2, \ldots, T$ do
7     $\rho_i = 1 - \frac{V_{goal}}{|V_i|}$;
```

Algorithm 5: ANOMALY-SNAPNETS

```
  Data: $\{G_1, G_2, \ldots, G_T\}$
  Result: $R$
1 $\rho_1, \rho_2, \ldots, \rho_T$ =SET-$\rho(G_1, G_2, \ldots, G_T)$;
2 For each $G_i$ coarsen with $\rho_i$ and get $G_i^c$ in parallel;
3 Compute feature $\widehat{\mathbf{F}}$ vector of segments in $\mathcal{S}$ in parallel;
4 Generate the segmentation graph $\mathsf{G_s}$;
5 $c^*$ = Parallel-LAYERED-ALP ($\mathsf{G_s}$, $\mathcal{G}$, $s$, $t$ );
```

on the structure of the graphs in the sequence. As a consequence, graph size may change over time. We build the following framework to solve the same problem using SNAPNETS:

(1) Summarize *Act-snapshots*. The size of *Act-snapshots* can be different. Hence, as we explained in Section 5 we compute the compression ratio of each graph according to Algorithm 4. Next, we summarize each graph, assuming all nodes have the same label, to get *C-graphs* with the same size.

(2) Find c^* as R. We follow the same procedure as SNAPNETS to find c^* as R.

Algorithm 5 indicates the anomaly detection algorithm using SNAPNETS.

Justification: The detected *Cut-points* by SNAPNETS is an accurate solution because SNAPNETS detects important time points in the *AS-Sequence* where the structure of graphs change which in turn can be considered as anomalies (or important events).

6 EXPERIMENTS

We design various experiments to evaluate SNAPNETS. We implemented SNAPNETS in Python. Our experiments were conducted on a 4 Xeon E7-4850 CPU with $512GB$ of $1066Mhz$ main memory and we released our code and datasets for research purposes[3].

6.1 Setup

We design experiments to answer the following questions:

Q1. Is the coarsened graph a good summary for our segmentation problem?
Q2. How is the quality of our feature set?
Q3. What is the difference between ALP and other path optimization algorithms?
Q4. What are the segmentations found by SNAPNETS? Do they discover useful and interesting patterns?
Q5. Can ANOMALY-SNAPNETS find important events and anomalies in dynamic graph sequences?
Q6. Is SNAPNETS scalable to run on real large datasets?

6.2 Datasets

For our experiments, we collected a number of real-world and synthetic datasets from various domains such as social and news media, epidemiology, autonomous system, and co-authorship network to evaluate SNAPNETS. See Table 4 for a summary description.

The ground truth segmentations in these datasets are non-trivial. They are induced from complex structural changes such as activation in different parts of the *Act-snapshots* (*AS Oregon* and *Higgs*), and varying role of active nodes e.g. change of active nodes centrality (*BA-degree*), or activation rate changes (*Portland*). Moreover, detecting the number of correct cut points is also a difficult task itself. In datasets without ground truth, we would like to find how memes/news were adopted by social media users (*IranElect* and *Memetracker*) and when the co-authorship relation on a specific topic changes over time (*DBLP*).

Dataset	#Nodes	#Edges	Timesteps	GT
BA-degree	500	4,900	240 units	✓
AS-PA	633	1,086	400 units	✓
AS-COM	4431	7,609	530 units	✓
AS-MIX	1899	3261	1000 units	✓
Higgs	456,626	14,855,843	7 days	✓
Portland	1,575,861	19,481,626	25 days	✓
Memetracker	960	5,001	165 days	
IranElect	126,915	5,589,083	30 days	
DBLP	369,855	1,109,452	13 years	
WorldCup	140,336	674,077	30 days	✓
Security	308,499	1,182,021	90 days	✓
EnronInc.	82,797	349,780	111 weeks	✓

Table 4: **Datasets details. (GT == Ground Truth)**

(1) BA-degree. We activate highest degree and then lowest degree nodes on a random Barabasi Albert graph [16].

(2) AS Oregon contains an Autonomous Systems (AS) peering information network collected from the Oregon router views[4]. We generate three simulations: *AS Oregon*-PA, *AS Oregon*-COM and *AS Oregon*-MIX.: (a) *AS-PA*: First we activate nodes with **P**referential **A**attachment process and then uniformly randomly. In the next two simulations, we created three copies of *AS Oregon* and connected them with bridge edges. We assume each copy is a community in the combined graph. (b) *AS-COM*: we connect 7 copies of the original network and combine them into a large network with 7 communities. We start activating 3 communities at different times with the same PA activation process. (c) *AS-MIX*: It is a combination of the above two. Each community gets activated at different time, and the activation in each community has two stages with different patterns (i. e. PA and random stages).

(3) Higgs [17] is a related tweets dataset (with the follower-followee network) when the Higgs particle was discovered. Between 1st and 7th July 2012 several announcements were made about the discovery of the Higgs particle. Nodes of the graph (i.e. twitter users) are active when they participate in related activities

3. http://github.com/SorourAmiri/SnapNETS

4. http://topology.eecs.umich.edu/data.html

(e.g. retweeting, replying). The ground truth cut is the official release date of the discovery.

(4) Portland. has a contact network among a synthetic population of Portland, and this dataset has been used in national smallpox modeling studies [18]. We simulate a chain shape diffusion and then a star shape one in a synthetic contact network of Portland[5].

(5) IranElect contains the tweets and the follower-followee network of Twitter [19], during the 2009 Iran election. Users are the nodes of the network and they are active if they post a related tweet. We want to find how the news were adopted by twitter users.

(6) DBLP is a co-authorship network [20] related to the topic "network". Authors are nodes, and edges show the co-authorship relations. An author is active if she co-authors a related paper with "network" or its derivations in its title. We want to find how the co-authorship of this topic develops over time.

(7) Memetracker. It is the who-copies-from-whom blog and website network [21]. We consider a blog/website as active if it adopts the phrase 'hey can I call you joe'. We want to find how the adoption pattern changes.

To evaluate ANOMALY-SNAPNETS we collect three real world temporal graph datasets. The ground truth events of all datasets are available [6].

(1) WorldCup It is the World Cup twitter dataset (Jun 12 - Jul 13, 2014). Nodes represent entities (i.e. URLs, hashtags, mentions) and edges show the co-mention relation [22].

(2) Security It is a twitter data of four months (May 2 - August 1, 2014) related to the security and terrorism. Nodes represent entities (i.e. URLs, hashtags, mentions) and edges show the co-mention relation [22].

(3)EnronInc. It is the Enron email communication network from Jan 2000 to March 2002. Nodes represent email addresses and edges shows the sent/received relations [22].

6.3 Baselines

To the best of our knowledge, there is no existing algorithm which has all the desired properties (Section 3.2). Hence, we adapt three alternative approaches that we also mentioned in Section 3.1 as baselines: time series, clustering, and graph summarization algorithms.

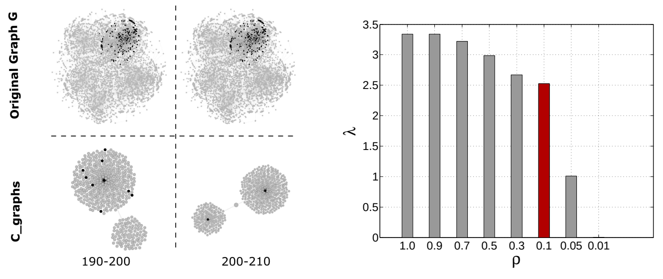
(1) DYNAMMO uses the spike of reconstruction errors to find change points in time series [1]. It has also been used in other time series problems [23]. We adapt it for our problem: Extract features in Table. 3 for each *Act-snapshot*. Then, use these features to form a time series and feed DYNAMMO to get the segmentation. The algorithm requires number of cut points as an input and we set it to the one SNAPNETS finds.

(2) K-MEANS finds segmentations based on an online "local" approach [2]. We make the same time series as in DYNAMMO, and feed it to K-MEANS. We report a new segment when a new cluster is detected. We consider it as the emergence of a new pattern and include the corresponding cut point in our final segmentation.

(3) VOG finds succinct descriptions of graphs in terms of common substructures [5], [4]. It does not work on labeled graphs. Hence, we extract active *sub-graph* of *Act-snapshots* and use VOG to find the 10 most important sub-structures. We output a segment if this set of sub-structure changes sufficiently (i.e is above a threshold which is set to be the one which gives the best results). Also, we use VOG as a baseline to evaluate the ANOMALY-SNAPNETS. In this case, the input of VOG is each snapshot of the temporal graphs.

(4) SELECT is an ensemble approach for anomaly mining. It leverage novel techniques to rank anomalies in temporal graphs. We take top $k = 5$ events as the detected anomalies and compere its result with ANOMALY-SNAPNETS.

Variations: Besides the above baselines, we design the following variations to test the functionality of each SNAPNETS's component. These variations are different from SNAPNETS only in one step.
(1) SN-ORIG extracts features from *Act-snapshots* and use the same $\mathsf{G_s}$ and ALP optimization to get the segmentation.
(2) SN-LP finds the **L**ongest **P**ath instead of ALP.
(3) SN-GREEDY greedily selects edges with the largest weight from s to t, instead of ALP.
Metrics. For datasets with ground truth, we measure the performance by calculating the F_1 score and precision of the set of detected cut-points with the ground-truth set (using the same methodology as [23]). For others, we perform case studies. We show how SNAPNETS reveals interesting patterns by matching the results with external news/events, and show how they are easily interpretable.



(a) (b)

Figure 4: **(a)** ***C-graphs*** **capture patterns that are not captured by the original graphs. (b)** λ **vs** ρ **shows** 0.1 **is the smallest** ρ **which maintains** λ**.**

6.4 Q1: Usefulness of C-graphs

We compare the performance of SNAPNETS with SN-ORIG to show the effectiveness of our summarization. See Table 5, SNAPNETS outperforms SN-ORIG: In most datasets SNAPNETS discovers the correct cut points ($F_1\ score = 1$), while SN-ORIG misses them. Hence clearly the *C-graphs* lead to better segmentation than the original *Act-snapshots*. Also, the *C-graphs* not only maintain the same pattern of the original *Act-snapshots*, but also discover important new patterns and help interpretation via the much simpler graph structure. For example, see figure 4a: the *C-graphs* of *AS-COM* (bottom row) capture clearly the pattern: first one community gets active, then two communities are active after the ground truth cut point 200. This pattern is hardly observable in the original *Act-snapshots* (top row).

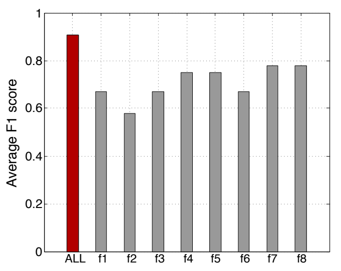
Also, we evaluate the largest eigenvalue of summary graphs of the *Memetracker* dataset with various compression ratio to show the effect of ρ on λ. Figure 4b, confirms our intuition behind selecting the ρ value in section 4.1 that 0.1 is a desired value for ρ: It is the smallest ρ which does not change the largest eigenvalue of the graph much.

5. http://ndssl.vbi.vt.edu/synthetic-data/
6. http://shebuti.com/SelectiveAnomalyEnsemble/

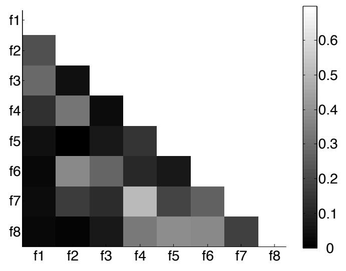
6.5 Q2: Quality of feature set

Here we show that the current feature set $\mathbf{F}$ we use is of high quality. We did the ablation test [24] to study the impact of each feature in the feature set $\mathbf{F}$. First we run SNAPNETS with the complete $\mathbf{F}$, we then remove one feature at a time from $\mathbf{F}$ to see the change of performance. In addition, we compute the correlation between each pair of features in $\mathbf{F}$.

The ablation test on the datasets with ground truth shows an average F_1 decrease of 0.2 (See Figure 5a). We also tried the same test on datasets without ground truth, and we observe unreasonable performance when we remove one of the features. For example, removing any of f_2, f_5, f_6 would lead to over-segmenting in *DBLP*; removing f_3 leads to an unreasonable cut point at the end of the sequence in *IranElect*. These results indicate the importance of each feature in $\mathbf{F}$. In addition, Figure 5b shows the correlation between pairs of features in $\mathbf{F}$ in all datasets (i.e. with or without ground-truth). It shows that most of the pairs has near zero correlation in at least one dataset and there is no pair with more than 0.5 correlation and on average the correlation is 0.17. These results show the high quality of $\mathbf{F}$.



(a)



(b)

Figure 5: **(a) The ablation test. (b)The correlation between features among all datasets. Darker color means less correlation.**

6.6 Q3: Finding the best path

Here, we evaluate the quality of ALP by comparing it to SN-LP and SN-GREEDY. See Table 5: SNAPNETS is much better than SN-LP and SN-GREEDY in all the datasets. We also check the number of segments found by SN-LP. As we expected (Section 4.3), it leads to over-segmentation. For example, in *Memetracker* and *IranElect* SN-LP detects cut point at every snapshot, whereas SNAPNETS gets the right answer every time.

6.7 Q4: Segmentation Results

Here we show the quality of the final segmentations found by SNAPNETS.

6.7.1 Quantitative analysis

We see in Table 5 that SNAPNETS has the best performance among baselines and variations of SNAPNETS. Note that all the baselines require some input parameters: such as the number of cut points (DYNAMMO), threshold for new cluster creation (K-MEANS), and difference threshold for outputting a cut point (VOG). We test a wide range of these input values and pick the ones that give the best results. Still, their performance is clearly worse.

BA-degree: SNAPNETS detects the ground truth segmentation while three other baselines do not perform as well. , when the activation starts to target low degree nodes instead of high degree nodes. As shown in Table 5, the F_1 *score* of SNAPNETS is 1. In comparison, the three other baselines do not correctly find the cut points where the pattern changes.

AS Oregon: SNAPNETS performs very well to differentiate between random and preferential attachment style activation (*AS-PA*) and we can accurately detect when different communities of the network get active (*AS-COM* and *AS-MIX*). Figure 6a visualizes the *C-graphs* in the c^*, and shows that our results are easily interpretable. In *AS-PA* dataset, SNAPNETS finds the ground truth cut point ($F_1\ score = 1$). In the more complicated *AS-COM* and *AS-MIX*, it also does very well, getting $F_1\ score$ of 0.67, 0.86 respectively. SNAPNETS also aids in interpretation: we visualize the *AS-MIX* segmentation in Figure 6a. We see clearly that the *C-graph* in the segment 2 captures the fact that there are two communities activated. As we activate more nodes in bridge, the active nodes in the two communities are merged to one node in the *C-graph* (segment 3). And finally when the third community gets activated, the *C-graph* again captures it as a new community (segment 4). The corresponding feature values of *C-graphs* also correctly reflect these changes (while the features from original graphs do not). Note that although the active subgraphs are similar in segments 1 and 3, and segments 2 and 4, the segments are actually different if we look at the whole graph including the inactive nodes (this difference is also seen in the feature values). These results show that SNAPNETS can detect non-trivial changes, including node-level, process-level or community-level changes.

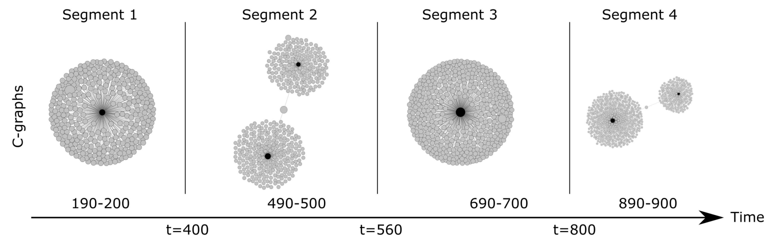
Higgs: SNAPNETS finds the exact ground truth Jul. 4 ($F_1 = 1$). Note that according to external news, there were rumors about the Higgs boson discovery from Jul. 2-4, these rumors make the ground truth harder to detect (for example VOG finds a cut point on Jul. 2 instead of Jul. 4). Further in the first segment, *C-graphs* have multiple near-clique structures of *active* nodes ($f_1 \simeq 0.9$, $f_5 = 0.2$) which demonstrates the existence of a rumor in the network (multiple small groups adopt the news). In comparison in the second segment, *C-graphs* become chain-like with many active nodes ($f_1 \simeq 0.02$, $f_5 = 0.9$). It suggests that many communities adopt the news (with nodes in the same community merged), and few bridge (inactive) nodes connect different communities, matching our expectation since the official announcement is evidently more influential.

Portland: SNAPNETS again detects the ground truth segments ($F_1 = 1$). Other baselines have worse performance except VOG and DYNAMMO since the change of the infection pattern in this dataset is visible in the active subgraphs: an infected chain first, and then many infected stars (as the disease goes viral). SNAPNETS shows its power in finding the pattern change in disease propagation.

6.7.2 Case studies

SNAPNETS finds meaningful segments in multiple datasets from various domains with varied patterns of evolution (both structurally and in labels) while none of the baselines perform as well (for example, VOG finds no cut-point in *DBLP*, K-MEANS essentially gives one at the end, and DYNAMMO finds no cut-point in *Memetracker*).

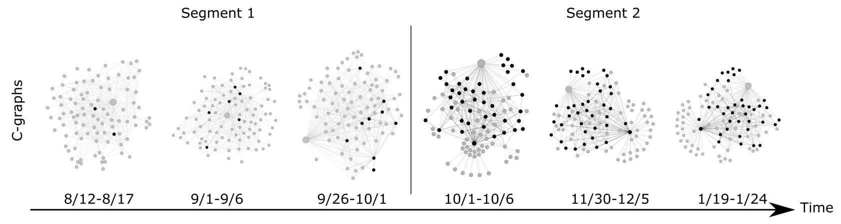
Memetracker: SNAPNETS finds a cut point on Oct 01, 2008, which matches the date of the televised debate between Joe Biden and Sarah Palin. In the first segment, *C-graphs* (Figure 6c) are close to the case when all nodes have the same label—suggesting



(a) Visualization of *C-graphs* in the segmentation for *AS Oregon*-MIX.

Feature	Seg. 1	Seg. 2	Seg. 3	Seg. 4
f1	0.32	0.28	0.90	0.76
f2	0.12	0.36	0.57	0.77
f3	0.50	0.65	0.83	0.87
f4	0	0	0	0
f5	0	1	0	1
f6	0.91	0	1	0
f7	0.26	0.14	0.87	0.42
f8	0	0	0	0

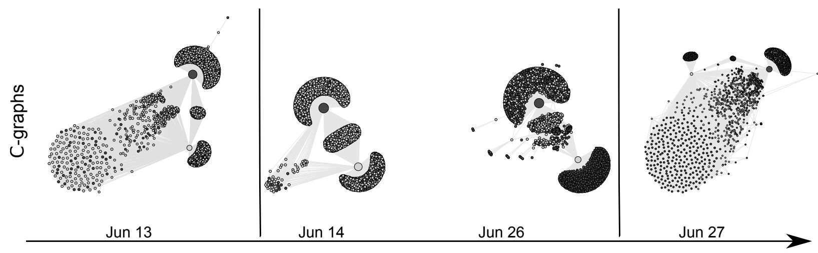
(b) The average feature values in each segment for *AS Oregon*-MIX.



(c) Visualization of *C-graphs* in the segmentation for *Memetracker*.

Feature	Seg. 1	Seg. 2
f1	0.93	0.06
f2	0.97	0.02
f3	0.09	0.97
f4	0.12	0.87
f5	0.10	0.95
f6	0.22	0.91
f7	0.45	0.76
f8	0.20	0.95

(d) The average feature values in each segment for *Memetracker*.



(e) Visualization of *C-graphs* in the segmentation for *IranElect*.

Feature	Seg. 1	Seg. 2	Seg. 3
f1	0.79	0.11	0.85
f2	0.56	0.04	0.81
f3	0.11	0.73	0.047
f4	0.92	0.14	0.82
f5	0.05	0.33	0.92
f6	0.19	0.70	0.01
f7	0.69	0.06	0.63
f8	0.21	0.20	0.89

(f) The average feature values in each segment for *IranElect*.

Figure 6: **Interpretation of the segmentation results found by SNAPNETS. (a), (c), (e) show the *C-graphs* for the segmentations found for *AS Oregon*-MIX, *Memetracker*, and *IranElect*. The vertical lines are the detected cut points. Black nodes are active and gray ones are inactive. (b), (d), and (f) show the average feature values in each of the segment for *AS Oregon*-MIX, *Memetracker*, and *IranElect*.**

Data w. GT	F_1 score						
	SNAPNETS	SN-ORIG	SN-LP	SN-GREEDY	DYNAMMO	K-MEANS	VOG
BA-degree	**1**	**1**	0.08	**1**	0	0.4	0.67
AS-PA	**1**	0	0.05	0.67	0	0.4	**1**
AS-COM	**0.67**	0	0.07	0.5	0.5	0.22	0
AS-MIX	**0.86**	0	0.07	0.57	0.32	0.4	0
Higgs	**1**	0	0.15	**1**	0	0.67	0
Portland	**1**	0	0.4	**1**	**1**	0.67	**1**

Table 5: F_1 **score of the segmentation detected by SNAPNETS, variations, and baselines on datasets with ground truth. SNAPNETS has the best performance among all competitors.**

that few nodes randomly got infected ($f_5 \simeq 0.1$, $f_8 \simeq 0.2$). Few active nodes in the first segment of Figure 6c confirms this fact and its because the meme is not viral yet. In the second segment, *C-graphs* are substantially sparser (i.e. f_2 dramatically dropped to 0.02) and contain large stars and leaf nodes with active centers. The size of the centers in the second segment of Figure 6c and average PageRank (f_6) in the *C-graphs* shows that important nodes such as "CNN" and "BBC" websites spread the meme to many nodes, and they are merged to form hubs.

***IranElect*:** We detect two cut points at Jun. 14 (presidential election) and Jun. 27 (vote recount). In the first segment (Figure 6e), when the election did not get much attention, multiple small *near-clique* structures (high $f_1 \simeq 0.8$ and low $f_2 \simeq 0.5$) of *C-graphs* shows small and highly connected groups of political activists were active. In the second segment, important nodes (e.g. news media such as "NYtimes") in different areas of the graph reported possible collusion in counting votes and they became active and formed multiple large stars of active nodes in *C-graphs* (low $f_1 \simeq 0.1$ and high $f_6 \simeq 0.7$). Finally, after the vote recount and news reports more people became interested and tweeted about the election. Hence, *C-graphs* became densely connected ($f_1 \simeq 0.85$, $f_2 \simeq 0.8$) because the remaining hub and bridge nodes became active and merged, while a few small/sparse subgraphs remained inactive.

***DBLP*:** We detect a cut point in the year 1997, which matches the publication time of the ground-breaking papers in network science [25], [26], [18], [16]. In the first segment, *C-graphs* are relatively connected, and few less-central nodes are active ($f_1 = 0.7$, $f_5 = 0.03$, $f_6 = 0.2$) which indicate the "network" topic did not get much attention. In the second segment, the number of active nodes in *C-graphs* increases dramatically, and significant nodes with high degree are among the active ones ($f_5 = 0.6$, $f_6 = 0.8$, $f_7 = 0.83$); which suggests influential people became interested in "network" and made it into a "hot topic"—high $f_8 = 0.9$ also confirms this fact that many nodes got active since one high degree neighbor of them is active.

6.8 Q5: Anomaly detection: Another application

We measure the quality (precision) of the segmentation by ANOMALY-SNAPNETS baseline in three real-world datasets to evaluate its performance in detecting anomalies and important events. Table 6 shows that ANOMALY-SNAPNETS has the best performance compared to baselines. Note that SELECT is specifically designed for anomaly detection and VOG needs a threshold as an input (Also, VOG did not finish and ran out of memory in two cases). However, we detect the anomaly automatically without any preset parameter or user interaction, showcasing SNAPNETS's usefulness.

WorldCup ANOMALY-SNAPNETS detects Jun 16, July 7 and July 10 as cut-points. Jun 16 is the match between Germany and Portugal, one of the most important games in group matches according to the ground truth. The July 7 and 10 cut-points distinguish the semi-final games and the final game.

Security ANOMALY-SNAPNETS accurately finds the important events at Jun 11 and July 29, 2014. Jun 11, 2014, matches the date when large regions of Iraq was seized by Iraqi militants and the attack of 'Boko Haram' in Nigeria and the following kidnapping of schoolgirls took place. July 29, 2014, matches the date of Ebola virus outbreak in Liberia.

EnronInc. We detect a cut point on Oct. 15 which matches the major anomaly when the Enron company announced a third-quarter loss. This is also recognized in SELECT [22] as a major anomaly.

Dataset	ANOMALY-SNAPNETS	VOG	SELECT
WorldCup	**1**	0	**1**
Security	**1**	-	0.8
EnronInc.	**1**	$\times$	0.8

Table 6: **The precision of detected anomalies by ANOMALY-SNAPNETS and other baselines. '-' means the method ran out of memory and '$\times$' means it did not finish after 4 days.**

6.9 Q6: Scalability

Each step of SNAPNETS is carefully designed to be memory efficient and sub-quadratic or near linear with respect to the size of data. We extract subgraphs with different sizes from *DBLP* (up to 10M edges), then we run SNAPNETS on them. Figure 7a shows that as expected SNAPNETS scales near-linearly with respect to the number of edges in the sequence. The running time of SNAPNETS shown in Figure 7a is reasonable and practical, especially considering that dynamic graph analysis are typically time consuming. For example, [4] takes more than 100 hours to run on a dataset with size $10M$ while SNAPNETS takes *less than an hour* (i.e. it is ~ 10 *times* faster).

Also, we evaluate the scalability of generating G_s and LAYERED-ALP in Figure 7b. We generate a series of $\mathcal{G}$ of *DBLP* data with various lengths (up to 2000 snapshots), then we run SNAPNETS on them. Figure 7b confirms that generating G_s and LAYERED-ALP are not the bottleneck of SNAPNETS. Also, it shows that our algorithm is much faster than the state-of-the-art algorithms.

Figure 7c shows the excellent speedup ($\sim$ 9X faster using 10 processors) we get after parallelizing the *Act-snapshots* summarization and feature extraction. Also, Figure 7d shows the speedup of making the segmentation graph in parallel ($\sim$ 8X using 10 processors). In summary, SNAPNETS makes it possible to process a large network sequences with many snapshots.

7 RELATED WORK

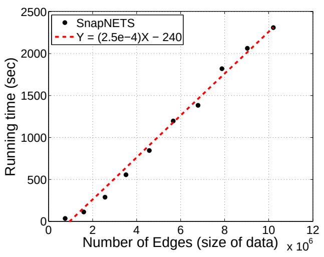
We briefly discuss some more related work (besides the most closely related ones in Sec. 3.1) in Time series and Dynamic graph analysis. Broadly speaking, unlike these methods, we maintain both the structural and label dependent properties of a graph.

Time series analysis: The research community has made great efforts in developing algorithms for different problems on time series data. These include algorithms for mining multivariate time series [27], summarizing time series with missing values [1], a generative analysis of time series [28], and many others. Among them, there is also much work about time series segmentation [29], [30], [31], rule and dimension discovery [32], [33], and other specialized algorithms like motion capture sequences mining [23].

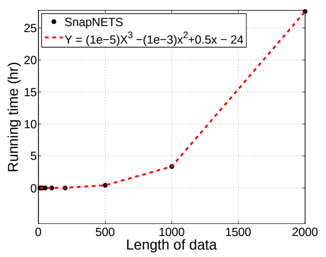
However, our problem is fundamentally different because we deal with sequences of *Act-snapshot*. And converting graphs (even without labels) to multivariate feature values while preserving the desired patterns is already a nontrivial task [34], [35]. Hence time series segmentation algorithms cannot be easily applied for our problem to get meaningful segmentations.

Graph summary and sparsification: It aims to find compact representations of graphs which maintain desired properties. The properties can be defined based on specific user queries [36], action logs [12], weights of nodes and edges [7], the drop of the leading eigenvalue [10], or more generally the encoding cost [37]. These algorithms help reducing the processing cost of large graphs, and maintain (sometimes amplify) the patterns. Unlike these methods our algorithm maintains the structural as well as label dependent properties of a graph.

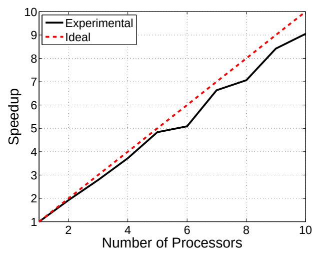
Dynamic graph analysis: As many natural networks evolve, this area is gaining much interest because of the evolutionary nature of many networks we see nowadays (see [38] for an overview). Many traditional machine learning tasks on static graphs have been extended to dynamic ones, such as the clustering problem [39], [40], classification problem [41], [42], link prediction [43], anomaly detection [11], trend mining [44]. There is also work finding time cut points according to the change of patterns in the dynamic graph. For plain dynamic graphs, [6], [45], [46], [47] detect the cut points when communities/partitions change suddenly, Ferlez et al. [6] use MDL principle to detect the cut points when communities in the evolving network change abruptly. Sun et al. [45] use MDL principles to find partition and segmentation of graph streams. Araujo et al. [48] use tensor decomposition with MDL to discover temporal communities in dynamic graphs. Their work is community-based while in our problem, we study the patterns in a more general way which is not restricted to communities or clusters.



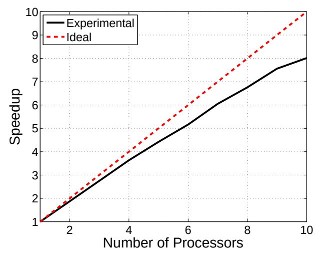
(a) Scalability W.R.T the size of $\mathcal{G}$



(b) Scalability W.R.T length of $\mathcal{G}$



(c) Parallel summarization speedup.



(d) Parallel $\mathsf{G_s}$ speedup.

Figure 7: **(a) and (b) shows the scalability of SNAPNETS with respect to the size and length of the $\mathcal{G}$. (c) and (b) show the speedup by parallelizing summarizing *Act-snapshots* and constructing $\mathsf{G_s}$.**

In contrast, we study the patterns in a more general way taking labels, not restricted to communities or clusters.

8 Discussion and Conclusions

We presented SNAPNETS, an intuitive and effective method to segment *AS-Sequences* with binary node labels and extended it to work with dynamic graph sequences as well. It is the *first* method to satisfy all the desired properties **P1**, **P2**, **P3**. It efficiently finds high-quality segmentations, detects anomalies and events and gives useful insights in diverse complex datasets. Also, we propose ANOMALY-SNAPNETS as an expansion of SNAPNETS to detect anomalies and important events in dynamic graph sequences. Finally we parallelize SNAPNETS and ANOMALY-SNAPNETS to accelerate the computations.

Patterns it finds: In short, SNAPNETS finds segmentations where adjacent segments have different characteristics of nodes with the same label i.e. the 'placement' and 'connection' of active/inactive nodes are different. This includes both structural (e.g. community/role/centrality) and rate changes. As a non-trivial example, we can detect both a random-vs-targeted activation and a faster-vs-slower one. Also, ANOMALY-SNAPNETS detects segments with different structural characteristics such as the density, the growing speed of the network, etc.

Global method: It is useful to note that SNAPNETS is a 'global' method and not simply a change-point detection method. We are not just looking for local changes; rather we track the 'total variation' using $\mathsf{G_s}$. Hence this allows to find important cut-points automatically and without any specification (which is useful for anomaly detection as well).

Flexibility: The SNAPNETS framework is very flexible, as our formulations are very general. Thanks to our careful design we easily expand our method to handle dynamic graphs with varying nodes and edges and propose ANOMALY-SNAPNETS. The eigenvalue characterization is general; similarly, the $\mathsf{G_s}$-ALP formulation should be also useful for other segmentation-like problems; and LAYERED-ALP can be of independent interest too.

Future work: Parallelizing LAYERED-ALP and extending our work to streaming graphs, and handling more general node/edge level features and partially observed graphs will be interesting. Also, analyzing the effect of using different compression ratio in ANOMALY-SNAPNETS is an interesting future work.

Acknowledgments. This paper is based on work partially supported by the National Science Foundation (IIS-1353346), the National Endowment for the Humanities (HG-229283-15), ORNL (Task Order 4000143330) and from the Maryland Procurement Office (H98230-14-C-0127), and a Facebook faculty gift. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the respective funding agencies.

References

[1] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos, "Dynammo: Mining and summarization of coevolving sequences with missing values," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 507–516.

[2] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[3] K. Henderson, T. Eliassi-Rad, C. Faloutsos, L. Akoglu, L. Li, K. Maruhashi, B. A. Prakash, and H. Tong, "Metric forensics: a multi-level approach for mining volatile graphs," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 163–172.

[4] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos, "Timecrunch: Interpretable dynamic graph summarization," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1055–1064.

[5] S. Shih, "Vog: summarizing and understanding large graphs." SIAM, 2014.

[6] J. Ferlez, C. Faloutsos, J. Leskovec, D. Mladenic, and M. Grobelnik, "Monitoring network evolution using mdl," in *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 2008, pp. 1328–1330.

[7] Q. Qu, S. Liu, C. S. Jensen, F. Zhu, and C. Faloutsos, "Interestingness-driven diffusion process summarization in dynamic networks," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2014, pp. 597–613.

[8] R. M. Anderson, R. M. May, and B. Anderson, *Infectious diseases of humans: dynamics and control*. Wiley Online Library, 1992.

[9] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 137–146.

[10] M. Purohit, B. A. Prakash, C. Kang, Y. Zhang, and V. Subrahmanian, "Fast influence-based coarsening for large networks," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 1296–1305.

[11] K. Henderson, T. Eliassi-Rad, C. Faloutsos, L. Akoglu, L. Li, K. Maruhashi, B. A. Prakash, and H. Tong, "Metric forensics: a multi-level approach for mining volatile graphs," in *KDD*, 2010.

[12] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen, "Sparsification of influence networks," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 529–537.

[13] B. A. Prakash, D. Chakrabarti, N. C. Valler, M. Faloutsos, and C. Faloutsos, "Threshold conditions for arbitrary cascade models on arbitrary networks," *Knowledge and information systems*, vol. 33, no. 3, pp. 549–575, 2012.

[14] G. Li, M. Semerci, B. Yener, and M. J. Zaki, "Effective graph classification based on topological and label attributes," *Statistical Analysis and Data Mining*, vol. 5, no. 4, pp. 265–283, 2012.

[15] D. Salvi, J. Zhou, J. Waggoner, and S. Wang, "Handwritten text segmentation using average longest path algorithm," in *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*. IEEE, 2013, pp. 505–512.

[16] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[17] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi, "The anatomy of a scientific rumor," *Scientific reports*, vol. 3, 2013.

[18] S. Eubank, H. Guclu, V. A. Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang, "Modelling disease outbreaks in realistic urban social networks," *Nature*, vol. 429, no. 6988, pp. 180–184, 2004.

[19] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 591–600.

[20] T. Lappas, E. Terzi, D. Gunopulos, and H. Mannila, "Finding effectors in social networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 1059–1068.

[21] J. Leskovec, L. Backstrom, and J. Kleinberg, "Meme-tracking and the dynamics of the news cycle," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 497–506.

[22] S. Rayana and L. Akoglu, "Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs," *SDM15*, vol. 17, 2015.

[23] Y. Matsubara, Y. Sakurai, and C. Faloutsos, "Autoplait: Automatic mining of co-evolving time sequences," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 193–204.

[24] S. Sundereisan, A. Bhadriraju, M. S. Khan, N. Ramakrishnan, and B. A. Prakash, "Sanstext: Classifying temporal topic dynamics of twitter cascades without tweet text," in *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*. IEEE, 2014, pp. 649–656.

[25] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-worldnetworks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[26] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *ACM SIGCOMM computer communication review*, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[27] I. Batal, D. Fradkin, J. Harrison, F. Moerchen, and M. Hauskrecht, "Mining recent temporal patterns for event detection in multivariate time series data," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 280–288.

[28] P. Wang, H. Wang, and W. Wang, "Finding semantics in time series," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 385–396.

[29] A. Samé and G. Govaert, "Online time series segmentation using temporal mixture models and bayesian model selection," vol. 1, pp. 602–605, 2012.

[30] X. C. Chen, K. Steinhaeuser, S. Boriah, S. Chatterjee, and V. Kumar, "Contextual time series change detection." in *SDM*. SIAM, 2013, pp. 503–511.

[31] M. Mendoza, B. Poblete, F. Bravo-Marquez, and D. Gayo-Avello, "Long-memory time series ensembles for concept shift detection," in *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. ACM, 2013, pp. 23–30.

[32] B. Hu, T. Rakthanmanon, Y. Hao, S. Evans, S. Lonardi, and E. Keogh, "Discovering the intrinsic cardinality and dimensionality of time series using mdl," in *2011 IEEE 11th International Conference on Data Mining*. IEEE, 2011, pp. 1086–1091.

[33] M. Shokoohi-Yekta, Y. Chen, B. Campana, B. Hu, J. Zakaria, and E. Keogh, "Discovery of meaningful rules in time series," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1085–1094.

[34] D. Koutra, J. T. Vogelstein, and C. Faloutsos, "D elta c on: A principled massive-graph similarity function," in *Proceedings of the SIAM International Conference in Data Mining. Society for Industrial and Applied Mathematics*. SIAM, 2013, pp. 162–170.

[35] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web graph similarity for anomaly detection," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 19–30, 2010.

[36] W. Fan, J. Li, X. Wang, and Y. Wu, "Query preserving graph compression," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 157–168.

[37] W. Liu, A. Kan, J. Chan, J. Bailey, C. Leckie, J. Pei, and R. Kotagiri, "On compressing weighted time-evolving graphs," in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 2319–2322.

[38] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, p. 10, 2014.

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

[40] M. Gupta, C. C. Aggarwal, J. Han, and Y. Sun, "Evolutionary clustering and analysis of bibliographic networks," in *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*. IEEE, 2011, pp. 63–70.

[41] C. C. Aggarwal and N. Li, "On node classification in dynamic content-based networks." in *SDM*. SIAM, 2011, pp. 355–366.

[42] İ. Güneş, Z. Çataltepe, and Ş. Gündüz-Öğüdücü, "Ga-tvrc-het: genetic algorithm enhanced time varying relational classifier for evolving heterogeneous networks," *Data Mining and Knowledge Discovery*, vol. 28, no. 3, pp. 670–701, 2014.

[43] P. Sarkar, D. Chakrabarti, and M. Jordan, "Nonparametric link prediction in dynamic networks," 2012.

[44] E. Desmier, M. Plantevit, C. Robardet, and J.-F. Boulicaut, "Trend mining in dynamic attributed graphs," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013, pp. 654–669.

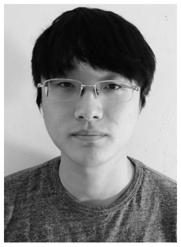
[45] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu, "Graphscope: parameter-free mining of large time-evolving graphs," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 687–696.

[46] C. C. Aggarwal and S. Y. Philip, "Online analysis of community evolution in data streams." in *SDM*. SIAM, 2005, pp. 56–67.

[47] K. S. Xu, M. Kliger, and A. O. Hero III, "Tracking communities in dynamic social networks," in *International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*. Springer, 2011, pp. 219–226.

[48] M. Araujo, S. Papadimitriou, S. Günnemann, C. Faloutsos, P. Basu, A. Swami, E. E. Papalexakis, and D. Koutra, "Com2: fast automatic discovery of temporal (comet) communities," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2014, pp. 271–283.

Sorour E. Amiri received the bachelors degree in computer engineering from Shahid Beheshti University and masters degrees in Algorithms and Computation from the University of Tehran, Iran. She is working toward the Ph.D. degree in the Department of Computer Science, Virginia Tech. Her current research interests include graph mining and social network analysis with a focus on segmentation of graph sequences and summarizing graphs. She has published papers in AAAI conference and ICDM workshops.



Liangzhe Chen received the bachelor's degree in computer science from Shanghai Jiao Tong University, China. He is working towards the PhD degree in the Department of Computer Science in Virginia Tech. His research interests include data mining, social network analysis, network vulnerability analysis, with a current focus on data sequence mining. He has published several papers in top conferences and journals such as ICDM, Data Mining and Knowledge Discovery Journal, AAAI.

B. Aditya Prakash is an Assistant Professor in the Computer Science Department at Virginia Tech. He graduated with a Ph.D. from the Computer Science Department at Carnegie Mellon University in 2012, and got his B.Tech (in CS) from the Indian Institute of Technology (IIT) – Bombay in 2007. He has published more than 50 refereed papers in major venues, holds two U.S. patents and has given three tutorials (SIGKDD 2016, VLDB 2012 and ECML/PKDD 2012) at leading conferences. His work has also received a best paper award and two best-of-conference selections (CIKM 2012, ICDM 2012, ICDM 2011) and multiple travel awards. His research interests include Data Mining, Applied Machine Learning and Databases, with emphasis on big-data problems in large real-world networks and time-series. His work has been funded through grants/gifts from the National Science Foundation (NSF), the Department of Energy (DoE), the National Security Agency (NSA), the National Endowment for Humanities (NEH) and from companies like Symantec. He received a Facebook Faculty Gift Award in 2015. He is also an affiliated faculty member at the Discovery Analytics Center at Virginia Tech. Aditya's homepage is at: http://www.cs.vt.edu/ badityap.