SnapNETS: Automatic Segmentation of Network Sequences with Node Labels

Sorour E. Amiri, Liangzhe Chen, B. Aditya Prakash
Department of Computer Science, Virginia Tech
Email: {esorour, liangzhe, badityap}@cs.vt.edu

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 is on plain graphs (and hence ignore the label dynamics) or fix parameters or require much feature engineering. Instead, 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.

Extensive experiments on several diverse real datasets show that it finds cut points matching ground-truth or meaningful external signals outperforming non-trivial baselines. We also show that SNAPNETS scales near-linearly with the size of the input.

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. It 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. We assume binary labels and can handle dynamic graphs with varying nodes and edges. 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:

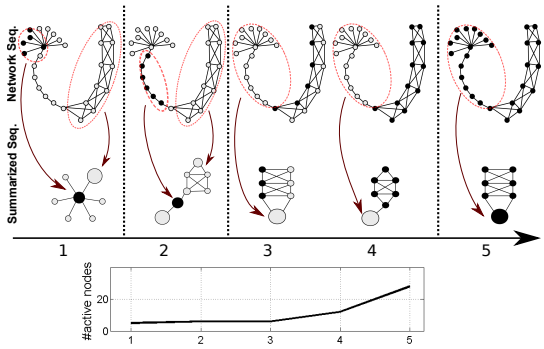


Figure 1: **TOY EXAMPLE: SNAPNETS automatically identifies four significant steps of the network sequence. The extracted time series (e.g. #active nodes) can not capture a proper segmentation. Gray nodes are inactive (i.e. label 0), and black nodes are active (i.e. label 1).**

PROBLEM 1: SEGMENTATION

***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, 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

Approaches / Properties	SNAPNETS	Feature eng. and time series (E.g. Li et al. 2009, Likas et al. 2003, Henderson et al. 2010)	Plain-graph-based (E.g. Shah et al. 2015, Koutra et al. 2014, Ferlez et al. 2008, Qu et al. 2014)
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.**

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).

Any algorithm should have these desired properties:

P1. Parameter-free: Find the best number of segments and segmentation without use of parameters such as change threshold and time window.

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

P3. Scalable: The method must be scalable (i.e. scales subquadratically 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. Most methods that we can adapt to solve this problem do not satisfy the above three properties—instead we propose SNAPNETS which does (Tab. 1 shows a brief comparison; more discussion in Sec. 2.1). 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. Finally, we also demonstrate SNAPNETS's usefulness via multiple experiments on diverse real-datasets.

The rest of the paper is organized in the standard way with first an overview and then details.

2 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, we assume that the network remains constant through time and we treat the problem as one with a series of graph snapshots: though our ideas can be easily used for other types of dynamic graphs, including with varying network structure (also shown in experiments).

Finally, we allow that nodes can even *switch* between labels freely (c.f. Fig. 1). This means we can handle both progressive/non-progressive scenarios: e.g., in the fundamental **S**usceptible-**I**nfected (SI) or **I**ndependent **C**ascade (IC) propagation models (where nodes once active, can not get inactive) and the 'flu-like' **S**usceptible-**I**nfected-**S**usceptible (SIS) model where infected nodes can get healthy again. 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_j is 1 if v_j 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 m is a partition of time interval $[1, T]$ with m time stamps i.e. $c = \{a_1, a_2, \ldots, a_m\}$ 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$_i$ $= \{v_j | l_j = 1\}$.*

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.

2.1 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 and the best choice of features may differ for different sequences (Henderson et al. 2010). Second, typical time series segmentation algorithms, which use "local" change detection, do not satisfy our desired properties. They usually need a threshold (Likas, Vlassis, and Verbeek 2003) (which usually depends on knowing the number of desired segments) to detect a change; or they fix *one* aggregation time period for the tracking (Li et al. 2009). 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* (Shah et al. 2015; Koutra et al. 2014; Qu et al. 2014). As we show in Fig. 1, using only the active subgraphs 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.

[1] Extending to multiple labels is interesting future work.

2.2 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 now compute the distance between any adjacent segments $w(s_{i,j}, s_{j,k})$ (each segment will contain sets of *Act-snapshots* G_i)? We want to use the entire graph (due to **P2**), while avoiding extracting complex features. Note that 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 an efficient novel algorithm LAYERED-ALP to find this path.

In short, we pursue 3 main goals: (1) *Summarize* G_i; (2) *Construct* $\mathsf{G_s}$ and (3) *Define and find the best segmentation*.

3 SnapNETs: Details

3.1 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 (Mathioudakis et al. 2011) 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 *no. 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 (Prakash et al. 2012) 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*. Thus, we define the summarization problem as follows,

Type	ID	Name
Structural	f_1	Largest eigenvalue of the adjacency matrix
	f_2	Number of edges
	f_3	Entropy of the edge weight distribution
	f_4	Average clustering coefficient
Label based	f_5	Number of active nodes
	f_6	Average PageRank of active nodes
	f_7	Average degree of active nodes
	f_8	Average degree of active neighbors of active nodes

Table 2: **Features extracted to represent each summarized *Act-snapshot* (i.e. *C-graph*).**

PROBLEM 2: *Act-snapshot* SUMMARIZATION

***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

$$\underset{|V_i^c|=\rho|V_i|}{\text{minimizes}} \; |\lambda_{G_i} - \lambda_{G_i^c}| \;\; \text{subject to} \;\; \textstyle\sum_{(a,b)\in E_i \text{ is merged}} |l_a - l_b| = 0$$

Here λ_G is the leading eigenvalue of graph G and l_a is the label of node a. 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 (Purohit et al. 2014)) whose goal is to maintain just 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 not allowing merging of node-pairs with different labels. This works very well in practice and gives near-linear running time. Note that a better algorithm for PROBLEM 2 will only improve our results. We use the same amount of coarsening ($\rho = 0.1$) as in (Purohit et al. 2014).

Fig. 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.

3.2 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 (Koutra et al. 2014). 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 exponential number of possible segmentations in polynomial space.

Feature extraction of C-graphs

Extracting features from G_i^c is 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 (Li et al. 2012) and eliminate correlated ones to get eight features for each G_i^c (See Tab. 2 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 (Li et al. 2012). 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 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 $d(s_{i,j}, s_{j,k}) = ||\widehat{\mathbf{F}}_{s_{i,j}} - \widehat{\mathbf{F}}_{s_{j,k}}||_2$.

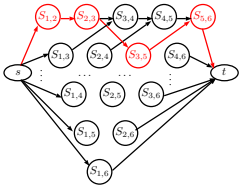


Figure 2

Fig. 2 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). Also, we need to compute the summary just once for each G_i, *not* for each segment in $\mathsf{G_s}$. We can compute the distance for every edge in $\mathsf{G_s}$ independently. Hence, we summarize *Act-snapshots* and construct the segmentation graph in *parallel*.

3.3 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 $p \in \mathcal{P}$.*

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.

$$c^* = \arg\max_{c \in \mathcal{P}} \frac{\sum_{s_{i,j}, s_{j,k} \in} w(e(s_{i,j}, s_{j,k}))}{|c|} \quad (1)$$

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 # nodes in the path (# segments).

An alternative 'parameter-free' optimization would have been the Longest Path (LP) problem: which will try to find the *longest* (heaviest) path in $\mathcal{P}$ (Eq. 1, 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 (Waggoner et al. 2013). In practice our observations confirm that LP contains unnecessary edges with low weight (see Sec. 4). Our ALP objective is intuitive and overcomes the disadvantage of LP. Fig. 2 shows the ALP for TOY EXAMPLE in red.

LAYERED-ALP ALP can be solved in poly. time on DAGs (recall $\mathsf{G_s}$ is a DAG)[2]. Current state-of-the-art algorithm (Waggoner et al. 2013) can solve PROBLEM 3 in $O(V_s^2.E_s)$. 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 we build a queue of layers of $\mathsf{G_s}$. Each layer i contains nodes which can reach t by i steps. When we iterate through layers, we maintain the weight ($P_i(v, t)$) of the heaviest path from v to t in i steps, and the next node of v in this path (κ_v^i). Alg. 2 shows the pseudo-code of LAYERED-ALP.

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

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

3.4 The complete algorithm

Alg. 1 shows the final pseudo code of SNAPNETS.

Lemma 4 SNAPNETS *has sub-quadratic time complexity $O(E \cdot \log E + \frac{E}{p})$, where E is the total number of edges in $\mathcal{G}$ and p is number of processors to parallelize constructing the segmentation graph $\mathsf{G_s}$.*

[2] ALP is NP-hard on general graphs

Algorithm 1: SNAPNETS

Data: *AS-Sequence* $\mathcal{G}$
Result: The optimal segmentation c^*
For each $G_i \in \mathcal{G}$ coarsen and get G_i^c once (Sec. 3.1);
Compute feature $\widehat{\mathbf{F}}$ vector of segments in $\mathcal{S}$ (Sec. 3.2);
Generate the segmentation graph $\mathsf{G_s}$ (Sec. 3.2);
c^* = LAYERED-ALP ($\mathsf{G_s}$, $\mathcal{G}$, s, t) (Sec. 3.3);

Algorithm 2: LAYERED-ALP

Data: $\mathsf{G_s}$, $\mathcal{G}$, s, t
Result: P_{alp}
Initialize Queue;
$Layer_0 = \{t\}$ and Queue.push($Layer_0$);
$Layer_1 = \{s_{i,j} | j = T\}$;
Queue.push($Layer_1$);
for k = 2 ***to*** T **do**
    $Layer_k = \{s_{i,j} | T - j + 1 \geq k\} \cup \{s\}$
    Queue.push($Layer_k$);
$Layer_{T+1} = s$ and Queue.push($Layer_{T+1}$);
LL = Queue.pop(), $\kappa_t^0 = \varnothing$;
while *Queue is not empty* **do**
    CL = Queue.pop() **for** $s_{i,j} \in$ *CL* **do**
        $\kappa_{s_{i,j}}^{CL} = \arg\max_{s_{j,k}} P_{LL}(s_{j,k}, t) + w(e(s_{i,j}, s_{j,k}))$;
        $P_{CL}(s_{i,j}, t) = P_{LL}(\kappa_{s_{i,j}}^{CL}, t) + w(e(s_{i,j}, \kappa_{s_{i,j}}^{CL}))$;
    If s is in CL then $lp_{CL} = P_{CL}(s, t)$;
    LL = CL;
$ALP = \arg\max(\frac{lp_1}{1}, \ldots, \frac{lp_T}{T})$;
Extract the P_{alp} using κ_s^T to κ_t^0;

4 Experiments

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

Datasets. We collected a number of datasets from various domains such as social and news media, epidemiology, autonomous system, and co-authorship network to evaluate SNAPNETS. See Tab. 3 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*[4] and *Higgs* (De Domenico et al. 2013)), and varying role of active nodes e.g. change of active nodes centrality (*BA-degree*), or activation rate changes (*Portland*[5]). 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* (Leskovec, Backstrom, and Kleinberg 2009)) and when the co-authorship relation on a specific topic changes over time (*DBLP* (Lappas et al. 2010)).

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	

Table 3: **Datasets details. (GT == Ground Truth)**

Baselines. To the best of our knowledge, there is no existing algorithm which has all the desired properties. Hence, we adapt two time series based algorithm, and one plain-graph-based algorithm as baselines. We show the details in Tab. 4.

Variations: We also designed the following three variations of SNAPNETS for comparison. *(1)* SN-ORIG extracts features from *Act-snapshots*. *(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 of the set of detected cut-points with the ground-truth set. 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.

4.1 Segmentation Results

We give representative results here.

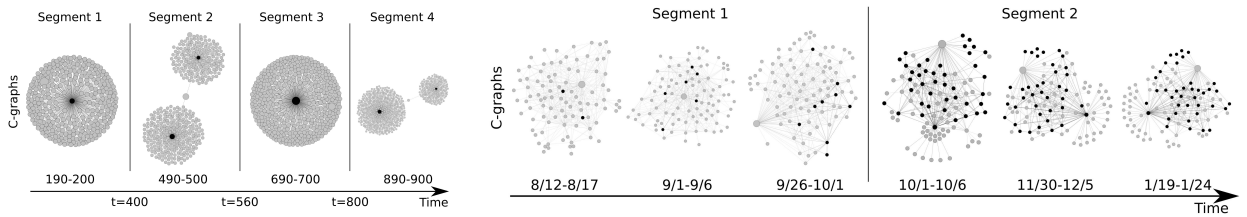
Quantitative analysis We see in Tab. 5 that SNAPNETS has the best performance among baselines and variations of SNAPNETS. Note that all the baselines require some input parameters: such as the number of cut points (DYNAMMO), threshold for new cluster creation (K-MEANS), and difference threshold for outputting 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.

***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*). Fig. 3a visualizes the *C-graphs* in the c^*, and shows that our results are easily interpretable.

***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).

***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.

[3] http://github.com/SorourAmiri/SnapNETS
[4] http://topology.eecs.umich.edu/data.html
[5] http://ndssl.vbi.vt.edu/synthetic-data/



(a) *AS-MIX* (b) *Memetracker*

Figure 3: **Visualization of *C-graphs* for the segmentations found by SNAPNETS for *AS-MIX* and *Memetracker*. The vertical lines are the detected cut points. Black nodes are active and gray ones are inactive.**

Baseline	Description
DYNAMMO (Li et al. 2009)	Construct time series using features in Tab. 2, then feed the time series and the no. of cut points (detected by SNAPNETS) to DYNAMMO, and use reconstruction errors to find change points.
K-MEANS (Likas et al. 2003)	Construct time series similarly as in DYNAMMO, then it finds segmentations based on an online "local" approach, and reports a segment when a new cluster is detected.
VOG (Koutra et al. 2014)	Extract active *sub-graph* (VOG does not work on labeled graphs) and use VOG to find the 10 most important sub-structures. When the set of important substructures changes sufficiently (above a threshold which is set to be the one with the best performance), we output a segment.

Table 4: **Baselines description**

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 GT.**

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* (Fig. 3b) are close to the case when all nodes have the same label—suggesting that few nodes randomly got infected ($f_5 \simeq 0.1$, $f_8 \simeq 0.2$). 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 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, 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 such as "NYtimes" in different areas of the graph 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, *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.

4.2 Scalability

We performed scalability tests, and as expected SNAPNETS scales near-linearly w.r.t the no. of edges in $\mathcal{G}$ and the running time is reasonable and practical—it is ~ 10 *times* faster than dynamic graph analysis methods such as (Shah et al. 2015). Also, we get excellent speed-ups ($\sim$ 9X faster using 10 processors) after parallelization.

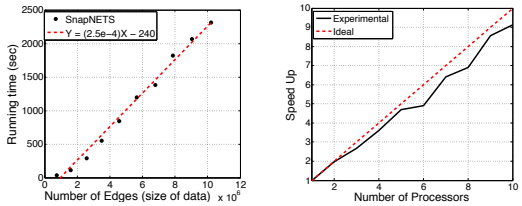


Figure 4: **(a) Scalability of SNAPNETS. (b) Speedup by parallelizing construction of G_s.**

5 Related Work

We discuss some related work (besides the most closely related ones in Sec. 2.1) in Time series and Dynamic graph analysis. Generally, unlike these methods, we maintain both the structural and label dependent properties of a graph.

Time series analysis: There has been much work for time-series (TS) data: including algorithms for mining multivariate TS, summarizing TS with missing values (Li et al. 2009), segmenting TS (Chen et al. 2013), rule and dimension discovery (Hu et al. 2011; Shokoohi-Yekta et al. 2015), and many others. However, our problem is fundamentally different because we deal with *AS-Sequence*; hence these algorithms cannot be easily applied for our task.

Dynamic graph analysis: As many natural networks evolve, this area is gaining interest: see (Aggarwal and Subbian 2014) for an overview. Many traditional machine learning tasks on static graphs have been extended to dynamic ones (Aggarwal and Li 2011; Sarkar, Chakrabarti, and Jordan 2012). For plain dynamic graphs, (Ferlez et al. 2008; Sun et al. 2007; Aggarwal and Philip 2005; Xu, Kliger, and Hero 2011) detect the cut points when communities/partitions change abruptly, while (Araujo et al. 2014) uses tensor decomposition to discover temporal communities. In contrast, we study the patterns in a more general way taking labels, not restricted to communities or clusters.

6 Discussion and Conclusions

We presented SNAPNETS, an intuitive and effective method to segment *AS-Sequences* with binary node labels. It is *first* method to satisfy all the desired properties **P1**, **P2**, **P3**. It efficiently finds high-quality segmentations, detects anomalies and gives useful insights in diverse complex datasets.

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.

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.

Flexibility: The SNAPNETS framework is very flexible, as our formulations are very general. 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. Adapting SNAPNETS to handle dynamic graphs with varying nodes and edges is useful as the next step. Also, extending our work to streaming and partially observed graphs, and handling more general node/edge level features will be interesting.

Acknowledgments. We would like to thank anonymous reviewers for their suggestions. 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

Aggarwal, C. C., and Li, N. 2011. On node classification in dynamic content-based networks. In *SDM*.

Aggarwal, C. C., and Philip, S. Y. 2005. Online analysis of community evolution in data streams. In *SDM*.

Aggarwal, C., and Subbian, K. 2014. Evolutionary network analysis: A survey. *ACM Comput. Surv.*

Araujo, M.; Papadimitriou, S.; Günnemann, S.; Faloutsos, C.; Basu, P.; Swami, A.; Papalexakis, E. E.; and Koutra, D. 2014. Com2: Fast automatic discovery of temporal ('comet') communities. In *PAKDD*.

Chen, X. C.; Steinhaeuser, K.; Boriah, S.; Chatterjee, S.; and Kumar, V. 2013. Contextual time series change detection. In *SDM*.

De Domenico, M.; Lima, A.; Mougel, P.; and Musolesi, M. 2013. The anatomy of a scientific rumor. *Scientific Reports*.

Ferlez, J.; Faloutsos, C.; Leskovec, J.; Mladenic, D.; and Grobelnik, M. 2008. Monitoring network evolution using mdl. In *ICDE*. IEEE.

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

Hu, B.; Rakthanmanon, T.; Hao, Y.; Evans, S.; Lonardi, S.; and Keogh, E. 2011. Discovering the intrinsic cardinality and dimensionality of time series using mdl. In *ICDM*. IEEE.

Koutra, D.; Kang, U.; Vreeken, J.; and Faloutsos, C. 2014. Vog: Summarizing and understanding large graphs. In *SDM*.

Lappas, T.; Terzi, E.; Gunopulos, D.; and Mannila, H. 2010. Finding effectors in social networks. In *KDD*.

Leskovec, J.; Backstrom, L.; and Kleinberg, J. 2009. Meme-tracking and the dynamics of the news cycle. In *KDD*.

Li, L.; McCann, J.; Pollard, N. S.; and Faloutsos, C. 2009. Dynammo: Mining and summarization of coevolving sequences with missing values. In *KDD*.

Li, G.; Semerci, M.; Yener, B.; and Zaki, M. J. 2012. Effective graph classification based on topological and label attributes. *Statistical Analysis and Data Mining*.

Likas, A.; Vlassis, N.; and Verbeek, J. J. 2003. The global k-means clustering algorithm. *Pattern recognition*.

Mathioudakis, M.; Bonchi, F.; Castillo, C.; Gionis, A.; and Ukkonen, A. 2011. Sparsification of influence networks. In *KDD*.

Prakash, B. A.; Chakrabarti, D.; Valler, N. C.; Faloutsos, M.; and Faloutsos, C. 2012. Threshold conditions for arbitrary cascade models on arbitrary networks. *KAIS*.

Purohit, M.; Prakash, B. A.; Kang, C.; Zhang, Y.; and Subrahmanian, V. 2014. Fast influence-based coarsening for large networks. In *KDD*.

Qu, Q.; Liu, S.; Jensen, C. S.; Zhu, F.; and Faloutsos, C. 2014. Interestingness-driven diffusion process summarization in dynamic networks. In *ECML/PKDD*.

Sarkar, P.; Chakrabarti, D.; and Jordan, M. I. 2012. Nonparametric link prediction in dynamic networks. In *ICML*.

Shah, N.; Koutra, D.; Zou, T.; Gallagher, B.; and Faloutsos, C. 2015. Timecrunch: Interpretable dynamic graph summarization. In *KDD*.

Shokoohi-Yekta, M.; Chen, Y.; Campana, B.; Hu, B.; Zakaria, J.; and Keogh, E. 2015. Discovery of meaningful rules in time series. In *KDD*. ACM.

Sun, J.; Faloutsos, C.; Papadimitriou, S.; and Yu, P. S. 2007. Graphscope: Parameter-free mining of large time-evolving graphs. KDD.

Waggoner, J.; Wang, S.; Salvi, D.; and Zhou, J. 2013. Handwritten text segmentation using average longest path algorithm. In *WACV*.

Xu, K. S.; Kliger, M.; and Hero, III, A. O. 2011. Tracking communities in dynamic social networks. In *SBP*. Springer.