ADAGE: A Framework for Generating Adaptable Intervals from Streaming Edges

Sucheta Soundarajan · Acar Tamersoy · Elias Khalil · Duen Horng Chau · Tina Eliassi-Rad · Brian Gallagher · Kevin Roundy

Received: date / Accepted: date

Abstract We study the problem of determining the proper aggregation granularity for time-evolving network data when edges are added to the network in a streaming fashion. Time-evolving (a.k.a. longitudinal or dynamic) networks are often used to study topics such as change detection, evolution of communities, or network growth. However, aggregation lengths are often somewhat arbitrary, chosen based on intuition or convenience. The same network may be aggregated per-day in one study, per-week in another study, and per-month in yet another. It is unclear whether these interval lengths are appropriate for the tasks being considered. We describe a novel algorithmic framework, called ADAGE, which systematically detects the appropriate variable-length intervals based on convergence of a network statistics (e.g., exponent of the degree distribution) or performance on a graph-mining task (e.g., true positive rate on belief propagation, BP). We apply ADAGE to 11 different network statistics and applications on 9 datasets from disparate domains. We observe that certain statistics consistently produce intervals suitable for various applications—e.g., aggregation lengths based on the exponent of the degree distribution are suitable for BP. In addition, we present 2 case studies—one on BP and another on network-similarity—which demonstrate the usefulness of ADAGE in practice. We observe that in some applications (e.g., BP for malware detection) shorter aggregation lengths produce better performance.

Keywords Longitudinal network analysis · graph mining · streaming edges

S. Soundarajan, T. Eliassi-Rad
Rutgers University
E-mail: {s.soundarajan, eliassi}@cs.rutgers.edu

A. Tamersoy, E. Khalil, D.H. Chau
Georgia Institute of Technology
E-mail: {tamersoy, lyes, polo}@gatech.edu

B. Gallagher
Lawrence Livermore National Laboratory
E-mail: bgallagher@llnl.gov

K. Roundy
Symantec Inc.
E-mail: kevin_roundy@symantec.com
1 Introduction

The study of longitudinal network data is important for understanding a variety of applications such as evolution of graphs and communities, detection of anomalous networks, etc. We address the problem of determining the proper interval for aggregating a sequence of edges into a network. Despite its importance, this problem has received very little attention from the research community. Existing approaches frequently select a single fixed-length interval, whose length is often arbitrarily selected. For instance, they group all edges that arrive during the same day into a single graph. These simple approaches have several potential problems. First, if the chosen interval length is too short, the resulting graph snapshots may not possess enough structure for meaningful analysis. In the worst case, such a graph might be a collection of unconnected edges (e.g., consider a tiny interval over Twitter Replies). Second, if the interval is too long, fine-grained changes in the graph structure may be obfuscated. Thus, algorithms for applications such as anomaly detection become less effective. Additionally, certain applications (such as belief propagation) can encounter performance degradation as graphs become denser (see Section 4.7). Third, data might stream at different rates throughout the timeline. Selecting a fixed-length interval for the entire timeline may simultaneously result in some intervals that are too short and others that are too long. Our goal is to find structurally mature network snapshots. More formally, we wish to identify the shortest time intervals such that the graph snapshots over those intervals possess the characteristics of real-world graphs. Such graph snapshots will then be suitable for graph-mining applications.

To solve the aforementioned problem, we introduce ADAGE, short for the Adaptable Graph Edge Interval Framework. ADAGE partitions a timeline of streaming graph edges into disjoint, variable-length intervals, each giving rise to a single graph snapshot. As its guide for what a structurally mature network should look like, ADAGE utilizes well-known characteristics of real-world graphs, such as power-law degree distributions with heavy tails [2]. To apply ADAGE, a user selects a graph statistic (such as the existence of a largest connected component) or an application of interest (such as belief propagation).\footnote{We consider seven popular network statistics as well as four important applications, including community detection and belief propagation. See Section 3.1 for details.} Given such a statistic (or application), the algorithm aggregates data until convergence is seen with respect to that statistic (or application). Figure 1 contains an example of this process, showing the values of a statistic (here the exponent of the degree distribution) for five different intervals at the various aggregation lengths. Once convergence has been achieved (see Section 3.2 for details), the algorithm stops its current iteration, sets aside the data that it has seen so far, and begins aggregating anew. We thus define a structurally mature snapshot as one in which adding additional data does not substantially change the value of the statistic being considered. It is important to note that we are looking for structurally mature snapshots, rather than attempting to find a sparse snapshot that represents the entirety of the timeline. One would certainly expect that statistics change substantially in different parts of the timeline; indeed, when studying network evolution of some graph statistic, one would hope that the statistic changes.
Example. Consider using ADAGE with the exponent of the degree distribution as the statistic of interest to partition a portion of the Facebook network consisting of wall posts, which are being collected on an hourly basis. This dataset is described in further detail in Section 4.1. We begin aggregating at hour 0, and continue aggregating until we see that the exponent of the degree distribution has converged. Figure 1 contains the results of this experiment. In this figure, dashed lines indicate the first five breakpoints found by ADAGE. The value of the curve is the exponent of the degree distribution at a given point in the aggregation. To demonstrate convergence, we show this value beyond the breakpoint. At the beginning of each interval, there was not enough data to calculate this exponent, and so the curves are not drawn there. We observe that the exponent is initially unstable. For the first interval, it begins at around -5.5, rises, drops back down, and then steadily increases to a value of approximately -1.2. In other intervals, the exponent begins at around -2.5, before again converging to approximately -1.2. These first few intervals range from 8 hours to 25 hours. Clearly, a fixed-length interval would not have worked well here. For this particular network and statistic, each snapshot has converged to roughly the same value of that statistic. This is not always the case. ADAGE cares about whether convergence has occurred, rather than the particular value.

Our experiments (detailed in Section 4) showcase (a) the necessity of short intervals (see Figure 2 for a prelude), (b) comparison of interval lengths produced by various statistics and applications, (c) running time analysis, (d) comparison with the most closely related method—i.e., DAPPER [8], (e) substitution of intervals, and (f) two case studies. In experiments (e), we design a study to determine which statistics and applications tend to produce similar interval lengths. An individual typically wishes to identify network snapshots for some specific application. For computationally intensive applications such as community detection, using ADAGE directly with this application may take a prohibitive amount of time. Results about interval similarity are thus valuable for two reasons: (1) they give insight into the

Fig. 1 Intervals obtained by ADAGE with Degree Distribution Exponent as the statistic on the Facebook1 network. Dashed lines indicate interval breakpoints. The curves show the value of the exponent of the degree distribution at various points in the aggregation. The curves do not start at the beginning of each interval because there is not enough data early in the interval to calculate the exponent of the degree distribution. See Section 1 for details.

Fig. 2 Results of belief propagation (BP) on the Symantec file × machine bipartite graph, with the goal of identifying malicious files and machines. For a fixed false positive rate, a higher true positive rate indicates better performance. (1) Shorter intervals can sometimes produce better results than longer intervals. (2) ADAGE offers a principled method for identifying intervals that are competitive with ad-hoc fixed-length intervals. More in Section 4.7.
network dynamic processes that give rise to different structural characteristics, and (2) they allow a user to use an efficiently calculated statistic to approximate intervals for a more computationally-intensive application. We also demonstrate that such substitutions are indeed effective. In experiments (f), we present a case study on belief propagation and another one on the problem of network similarity on longitudinal network data, in which one wishes to quantify the structural similarity between snapshots. Currently, selecting the appropriate interval lengths for these applications is ad-hoc, and ADAGE provides a systematics solution to this problem.

Our major contributions are as follows. (1) We present the first large-scale study—across a variety of network statistics, graph-mining tasks, and datasets—for partitioning a timeline of streaming network data into variable-length intervals. (2) We introduce ADAGE, a flexible online framework that can accommodate a variety of statistics and applications to produce structurally mature graph snapshots. (3) We show that for certain real-world applications (such as malware detection with belief propagation), snapshots aggregated over shorter time intervals can be preferable to snapshots aggregated over longer time intervals; thus one should not simply use a long, fixed-length interval to generate snapshots. (4) We show that certain pairs of statistics and applications give rise to similar intervals when used in ADAGE. These similarities are useful because intervals produced by ADAGE with computationally efficient statistics may be appropriate substitutes for computationally intensive applications. These results also give insights into network dynamics. (5) We demonstrate two use cases of ADAGE: belief propagation and network similarity.

The outline of the paper is as follows. In Section 2, we discuss in more detail the motivation and reasoning behind our work. Sections 3, 4, and 5, respectively, describe the ADAGE framework, our extensive experiments, and a discussion of the results. Section 6 presents some related work. We conclude in Section 7.

2 Background

Our work on ADAGE was inspired by three observations. Observation 1. Graph mining algorithms require their input graphs to possess some amount of structure (such as a large connected component, some preferentially attached nodes, etc). This observation was motivated by earlier work in [20], which studied the network similarity problem. It was observed that various network similarity methods exhibited a certain type of structured behavior on cross-sectional network data, but failed to exhibit this same behavior on longitudinal network data aggregated over a small time period. However, when these longitudinal datasets were aggregated over a larger time interval, the similarity methods exhibited their original, structured behavior. The authors hypothesized, and deeper analysis of the graphs revealed, that the short snapshots lacked meaningful structure (e.g., the graph was a collection of unconnected edges), and concluded that the data must be aggregated over some longer interval. In [20], the authors did not address what interval lengths should be used. Observation 2. For many reasons, one should use as short an interval as possible to produce a graph snapshot possessing the necessary structure. First, if one is given a finite timeline and wishes to understand network change, it makes sense to obtain as many structured snapshots as possible
within that timeline. This allows for more fine-grained understanding of graph dynamics. Moreover, some applications (like belief propagation) can perform poorly on dense graphs (details in Section 4.7). **Observation 3.** Intervals should be of variable lengths. Data can stream at very different rates during the observation timeline. For example, in the famed Enron Email dataset [9], some days contain tens of emails, while others contain hundreds. A fixed length interval would not be suitable in such cases.

To address these problems, we call on the rich history within the complex networks research community of studying patterns in real-world networks. For example, it is well known that real networks possess a large connected component, exhibit a power-law degree distribution (with heavy tail), have high clustering, and contain community structure. We postulate that a network exhibits structure (i.e., it is structurally mature) when it displays the patterns that we expect to see in a real-world network of that type.

## 3 Proposed Method: ADAGE

ADAGE, short for **Adaptable Graph Edge Interval Framework**, is a flexible online framework for aggregating streaming edge data into structurally mature graph snapshots. Specifically, given a graph statistic (a.k.a. feature or characteristic such as exponent of the degree distribution), ADAGE aggregates the streaming edges into a graph until the value of the chosen statistic has converged. Note that ADAGE seeks stability with respect to a statistic, rather than a specific value of that statistic.

ADAGE has several important strengths. It is **simple**, allowing for easy implementation and adoption to real-world problems. It is **flexible**, and can be tailored for any network statistic or graph-mining application. Section 3.1 presents the seven graph statistics and four applications considered in this paper. It is **efficient**, easily accommodating statistic approximation through sampling, calculation of statistic values distributed over multiple processors, or modification of convergence parameters.

Formally, ADAGE is defined as follows. The input to ADAGE consists of a function \( f(G) \) (which takes a graph as input and outputs a value of a statistic or application), a lookahead value \( L \), a threshold \( t \), and a streaming sequence of graph edges \( E_1, E_2, \ldots \) (where each \( E_i \) represents the set of edges arriving at time \( i \)).\(^2\) Given an initial starting point \( T_{\text{init}} \), ADAGE outputs a single breakpoint \( T_1 \) after \( T_0 \), so that all data between times \( T_0 \) and \( T_1 \) can be aggregated into a single graph snapshot. Through iteration, this can be used to partition a timeline (i.e., produce multiple breakpoints).

Let \( T_{\text{init}} \) be the point at which we begin aggregating the graph data. In the first iteration, \( T_{\text{init}} = 1 \). Then for \( T_i \geq T_{\text{init}} \), define \( G_{\text{init},i} \) to be the graph formed by aggregating all graph data between times \( T_{\text{init}} \) and \( T_i \). In this way, we obtain a sequence of graphs beginning with \( G_{\text{init},\text{init}} \), and by applying function \( f \) to each of these graphs, we obtain a sequence of values \( r_{\text{init}}, r_{\text{init}+1}, \ldots \), where each \( r_i = f(G_{\text{init},i}) \). We say that this sequence converges at time \( T_i \) if:

\(^2\) Here, we assume that this sequence is discretized. That is, multiple edges appear at the same time. However, this assumption is not necessary for use of ADAGE.
\[ \forall j \in [i, i + (L \times (i - init))] : r_j \in [(1 - t) \times r_i, (1 + t) \times r_i] \] (1)

Recall that \( t \) and \( L \) are the threshold and lookahead parameters. This definition assumes that all values are positive, and can be modified if any values are negative. The lookahead period is not \( L \) itself, but is \( L \) times the length of the interval seen so far. Longer intervals thus require longer lookahead periods. We discuss convergence issues and a guide to selecting values for \( t \) and \( L \) in Sections 3.2 and 3.3.

Given \( T_{init} \), once we find convergence at time \( T_i \), we form a graph snapshot by aggregating all data in that interval. The process then begins anew at time \( T_{i+1} \). If one has the entire data stream at hand, and wishes to run ADAGE offline, then this convergence rule can be modified by eliminating the lookahead \( L \) and requiring convergence until the end of the data stream. ADAGE can also be modified to require convergence of multiple statistics or applications simultaneously.

3.1 Graph Statistics and Applications

We consider seven graph statistics. Selection of a statistic set for aggregating a particular dataset requires some understanding of that dataset’s structure and desired properties. For example, if one is considering a bipartite graph, then statistics that rely on triangle counts are inappropriate.

**Exponent of Degree Distribution (DegDist):** Many real-world networks exhibit a power-law degree distribution (with a heavy tail) [2]. This characteristic is likely difficult to observe on a graph aggregated over a very short period of time. To calculate this value, we find the degree distribution of the network and then consider its log-log plot. We then find the slope of the best-fit line of this plot to estimate the exponent of the degree distribution.

**Exponent of Triangle Count Distribution (TriCount):** For many real-world networks, the distribution defined by \( f(x) \) equalling the number of nodes involved in \( x \)-many triangles also follows a power-law [23]. As with DegDist, we calculate the exponent of this power-law distribution.

**Exponent of Mean Number of Triangles for a Given Degree (DegTri):** We consider the distribution defined by calculating the average number of triangles in which nodes of each degree are involved. This distribution is also known to often follow a power-law, and as with DegDist and TriCount, we calculate the exponent of this distribution [23].

**Clustering Coefficient (CC):** Social networks tend to exhibit a high degree of clustering. That is, triangles exist in much higher numbers than one would expect in a random graph [24]. The clustering coefficient of a single node \( a \) is the fraction of triads \((a, b), (a, c)\) for which edge \((b, c)\) exists in the graph. For the CC statistic, we calculate the clustering coefficient of the graph, which is defined as the average of all of the nodes’ clustering coefficients.

**Largest Connected Component (LCC):** Many graphs ranging from random graphs to real-world social networks are known to possess a giant connected component, to which the vast majority of nodes belong [6]. When we consider longitudinal graph data, we would expect to see that at early time-points, the giant connected component does not yet exist. However, as the graph densifies,
this giant component will start to form. In this statistic, we measure the fraction of nodes in the largest connected component.

**Principal Eigenvalue (Eigs):** The eigenvalues of a graph carry information about the graph’s capacity. In this statistic, we calculate the largest eigenvalue of the adjacency matrix of the graph. Note that we expect this to be a poor statistic for obtaining convergence, as the principal eigenvalue will tend to increase as the number of edges increases. In fact, we see that on several datasets, it does not converge, but we include it for comparison.

**Effective Diameter (EffDiam):** Social networks also frequently exhibit a low diameter [24]. We would expect that this low diameter only emerges with sufficient data. In this statistic, we calculate the effective diameter of the graph, which is defined as the smallest value $D$ such that at least $90\%$ of node-pairs are within $D$ steps of one another [17].

We also consider four graph applications, which correspond to specific problems rather than general graph characteristics. Like with the statistics, each application returns a single value corresponding to a graph snapshot.

**Community Structure Stability (Comms):** Because real-world networks often display community structure, we calculate the stability of community structure between time steps [13]. We use the Louvain algorithm for modularity optimization to partition the set of nodes into groups exhibiting high internal density [5]. We define community stability from time step $T-1$ to time step $T$ as follows. Suppose the Louvain algorithm finds a set of communities $C = \{c_1, c_2, \ldots, c_n\}$ in the graph obtained by aggregating through time $T-1$, and found communities $C' = \{c'_1, c'_2, \ldots, c'_m\}$ in the graph obtained by aggregating through time $T$. Here, each $c_i$ and $c'_i$ is a set of nodes. We then use a version of the F1 score to calculate the similarity between these two sets. For each community $c_i \in C$, we find the greatest Jaccard similarity between $c_i$ and a community $c'_j$ in $C'$. We then define precision to be the average of these Jaccard similarities over all communities $c_i \in C$. To calculate recall, we do the reverse. The F1 score, or similarity, between the sets $C$ and $C'$ is then the harmonic mean of precision and recall. Intuitively, if the Louvain algorithm produces very similar partitionings in both graphs, then the F1 score will be high.

**PageRank Stability (PR):** One might wish to identify important nodes in a network. Here, we use the PageRank algorithm to identify the 100 most highly ranked nodes in a network [7]. Suppose $P$ is the set of the 100-highest PageRank nodes from the graph obtained by aggregating through time $T-1$, and that $P'$ is the set of the 100-highest PageRank nodes from the graph obtained by aggregating through time $T$. Then we define the stability of this set by calculating the Jaccard similarity between sets $P$ and $P'$.

**Degree Centrality Stability (Degs):** This application is similar to the PR application above, except that we rank nodes using their degree centrality rather than their PageRank scores.

**Belief Propagation (BP):** This application is valuable for networks containing positive and negative labels on the nodes. The BP algorithm iteratively infers unknown labels by using the known labels. BP cannot produce reasonable results unless the graph possesses some amount of structure that allows for such propagation. In this application, we calculate the true positive rate at a fixed false positive rate.


Table 1 Median values for interval lengths (in days) on the Enron Emails dataset using the DegDist statistic, with varying threshold and lookahead parameters. For some combinations, no intervals were found. See text for the explanation. Similar lookahead and threshold values were suitable on other networks, so we opt to use a lookahead of 0.1 and a threshold of 0.1 in our experiments.

<table>
<thead>
<tr>
<th></th>
<th>L = 0.1</th>
<th>L = 0.3</th>
<th>L = 0.5</th>
<th>L = 0.7</th>
<th>L = 0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 0.01</td>
<td>47.5</td>
<td>61</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>t = 0.05</td>
<td>8</td>
<td>8</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>t = 0.1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>none</td>
</tr>
<tr>
<td>t = 0.2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t = 0.3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3.2 Selection of Convergence Rule and Parameter Study

We use a parameter study to determine the values of the two convergence parameters described in Equation 1. Specifically, parameters $t$ (for threshold) and $L$ (for lookahead) can be any non-negative real value. Equation 1 allows a certain amount of variation in the values seen during the lookahead period. This variation is governed by the threshold parameter $t$. A smaller value for $t$ indicates that the permitted variation in the statistic value is low; thus, resulting in longer intervals (i.e., convergence is achieved more slowly). A smaller value for the lookahead parameter $L$ indicates that the statistic need not converge for a long period to be considered stable; thus, resulting in shorter intervals (i.e., convergence is achieved more quickly).

We report results of a parameter study, where we considered a variety of lookahead $L$ and threshold $t$ values, to determine appropriate values for them. We consider thresholds $t \in \{0.01, 0.05, 0.1, 0.2, 0.3\}$ and lookahead fractions $L \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Using each $(t, L)$ pair, we apply ADAGE to partition a timeline. We conducted this study on a variety of network datasets (described in the next section) and obtained similar results on each network. Table 1 contains the median interval length (in days) for each of these parameter combinations on the Enron Email dataset. We observed similar results on the other networks.

Thresholds $t$ of less than 0.1 produce very large intervals (indicating that convergence is slow). For some low thresholds and high lookahead fractions, ADAGE is unable to find a single interval over the entire timeline. In contrast, thresholds greater than 0.1 do not typically produce much shorter intervals than $t = 0.1$. Thus, we use $t = 0.1$. As for the lookahead fraction $L$, we observe that the value of $L$ does not have a significant impact on the interval length. Thus, we choose to use $L = 0.1$ (which is the smallest value considered). Recall that the lookahead period is not $L$ itself, but is rather $L$ times the length of the interval seen so far. Therefore, $L = 0.1$ means that the lookahead period is 10% of the interval length observed so far.\(^3\)

\(^3\) Because some intervals may be very short, we always require a minimum lookahead of 10 time-steps.
Assumptions. In its current form, ADAGE makes two assumptions. First, it assumes that the observed stream of edges is discretized in time. For example, the edge stream of an IP×IP communication network is discretized in seconds. ADAGE aggregates the edges per seconds to produce graph snapshots observed in variable-length intervals. The final ADAGE intervals are larger than a second. Clearly, if the initial intervals (i.e., the time discretization of the observed edge stream) is too large, then ADAGE might simply output each of these initial intervals as a structurally mature graph snapshot, while a shorter snapshot might have been obtained with smaller initial intervals. Second, ADAGE assumes that once an edge has been added to a snapshot, it is not removed for the remainder of that interval. If suitable for one’s data and application, this can be easily modified by simply removing deleted edges from the aggregation.

Convergence Guarantees. Suppose ADAGE begins its aggregation at time $T_0$. Let $s_i$ be the value of statistic $S$ for the graph snapshot containing data from time $T_0$ to $T_i$. Then, ADAGE is guaranteed to converge for statistic $S$ if there exists a time $T_j$ such that the following conditions hold:

1. For all $T_k$ greater than $T_j$, one of the following holds:
   (a) $s_k \geq s_{k+1}$ and statistic $S$ is bounded from below; OR
   (b) $s_k \leq s_{k+1}$ and statistic $S$ is bounded from above.
2. The set $\{s_k : k \geq j\}$ is of finite size.

Proof: Without loss of generality, suppose $S$ satisfies conditions 1(a) and (2). That is, $S$ is monotonically non-increasing after time $T_j$, and $S$ is bounded from below. Suppose that $S$ has not converged by time $T_j$. Let $l_j$ be the length of the lookahead period for interval $[T_0, T_j]$. Because $S$ has not converged at time $T_j$, this means that for some $k$ in the interval $[T_{j+1}, T_j + l_j]$, $s_k$ has a value that is different from $s_j$. Furthermore, $s_k < s_j$ because $S$ is non-increasing after time $T_j$. Repeating this argument, either $S$ has converged by time $T_k$, or there is some $k'$ in the interval $[T_{k+1}, T_k + l_k]$ with $s_{k'} < s_k$ (where $l_k$ is the length of the lookahead period for interval $[T_0, T_k]$). By condition 2 above, for $i > j$, $s_i$ can only take finitely many possible values. Thus, at some point, this argument must terminate because $S$ has reached its smallest possible value. ADAGE must converge at this point, if it has not yet converged. □

ADAGE is guaranteed to converge with any one of the seven statistics discussed earlier under the slightly idealized scenario in which we know the number of possible nodes, $N$, in the graph at the beginning of the aggregation period, and the assumption that once an edge is observed in an interval it is not removed. Consider the LCC statistic for example. The LCC statistic is monotonically non-decreasing, because its size can only increase. However, it cannot be greater than 1, so it is bounded from above. In addition, it can only take finitely many values, each of the form $\frac{i}{N}$, where $i$ is an integer between 0 and $N$.

Footnote: For any node $u$ in the graph, until node $u$ appears in the data stream as part of an edge, it exists as a singleton in the network, unconnected to any other nodes.
Extension 1. For some applications, one might wish to identify overlapping intervals. With overlapping intervals, one is guaranteed that consecutive graph snapshots have some structural overlap with one another, which is useful for cases when one wishes to understand local structural evolution or change. ADAGE can be easily modified to identify overlapping intervals. Recall that each iteration of ADAGE takes as input a single starting point and then outputs a single ending point. The implementation described above repeatedly applies this process to partition a timeline by setting each interval’s starting point to be the previous interval’s ending point. Overlapping intervals can be found in either of two ways. First, if a user provides a set of starting points that are sufficiently close to one another, then the resulting intervals will overlap one another. Second, if a user does not wish to specify starting points, ADAGE can expand the partitions that it originally found into overlapping intervals. The process is as follows: (a) use ADAGE to partition the timeline into disjoint intervals; and (b) extend each interval $[T_i, T_j]$. The extension is as follows. Because ADAGE uses a lookahead, the graph statistic or application under consideration remains stable for some amount of time after time $T_j$. Suppose that the lookahead extends to time $T_l$, which occurs after time $T_j$. The statistic or application remains converged through at least time $T_l$, but one can continue aggregating beyond that point until convergence is lost at time $T_k$. Interval $[T_i, T_j]$ can then be expanded into interval $[T_i, T_k]$, with the next interval starting at $T_j$ as before (recall that $T_j < T_k$). By applying this method to each interval, one obtains a series of overlapping intervals.

Extension 2. In this work, we only consider simple, undirected graphs. That is, once an edge has been seen in one interval, we ignore further occurrences of that edge in the same interval. ADAGE can easily accommodate directed or weighted graphs, as well as multi-graphs, as long as the statistic or application being used can be modified for such graphs (e.g., one can easily modify the DegDist statistic for use on directed graphs).

4 Experiments

This section presents our experiments on (a) the necessity of short intervals, (b) comparison of interval lengths produced by various statistics and applications, (c) running time analysis, (d) comparison with the most closely related method—i.e., DAPPER [8], (e) substitution of intervals, and (f) two case studies.

When partitioning a stream of edges into graph snapshots, there is no single “optimal” or “correct” interval length that will be suitable for all applications. For instance, the amount of structure needed for community detection may be very different from the amount needed for belief propagation. Because we lack a “ground truth” for the optimal or correct interval length, we validate our results by demonstrating that the intervals found by ADAGE are appropriate for different purposes. For example, we show that specific statistics, when used in the ADAGE framework, produce intervals that are similar to those produced by specific applications, both in terms of interval length as well as application performance (such as true positive rate for belief propagation). This type of result is useful because if a computationally efficient statistic produces interval lengths that are similar to those produced by some computationally intensive application, a user interested
in that application can use the cheap statistic to quickly find intervals suitable for his/her application, without having to perform the expensive computations required to use the application directly with ADAGE.

4.1 Datasets

We consider eight longitudinal network datasets from a variety of domains. For purposes of efficiency, we have discretized each of these network datasets into streams. Each stream element is a collection of edges, which are considered to have appeared at the same time. For each dataset, we select some number of random starting points from the timeline to begin ADAGE’s aggregation. The number of such starting points is dataset specific and is listed below.

**Facebook1 (FB1)** and **Facebook2 (FB2):** FB1 and FB2 are two portions of the Facebook network. The dataset we consider consists of wall posts made from March 2013 and March 2012, respectively, gathered by the MyPageKeeper Facebook app. Each node in this graph represents a Facebook account, and two nodes are connected if one of the nodes posted on the other’s Facebook Wall during the time period in question via some app. The stream elements here are the collection of edges observed per hour. Over the entire timeline, FB1 contains 754,995 nodes and 887,091 edges, and FB2 contains 865,103 nodes and 1,077,872 edges. For each of these datasets, we randomly select 100 starting points.

**Yahoo!:** This dataset consists of IM communications during April 2008. The stream elements here are collections of edges observed per day [25]. This dataset contains 100,000 nodes and 587,963 distinct edges. Because Yahoo contains only 28 days, we select 10 random starting points.

**Enron:** This is an email network [9]. It contains 84,429 nodes and 325,564 edges. The stream elements here are collections of edges observed per day. We select 100 random starting points from the Enron network.

**Blogs:** This is a graph of hyperlinks between blog posts, spanning 80 days [18]. It has 27,676 nodes and 126,227 edges. Its stream elements are collections of edges observed per day. We select 20 random starting points from Blogs.

**HEP:** This network contains paper citations from the Arxiv High Energy Physics section [12]. It contains papers from the years 1992 to 2002. HEP has 30,565 nodes and 346,849 edges. The stream elements here are collections of edges observed per year. We select 100 random starting points from HEP.

**IP:** This dataset contains IP-to-IP communications spanning one hour. It has 3,317 nodes and 9,637 edges. Its stream elements are collections of edges observed per second. We select 100 random starting points from IP.

**Symantec:** This dataset was obtained from Symantec’s WINE environment [11, 19]. It is a bipartite graph of machines and files. A file is linked to a machine if the file appeared on that machine. The dataset covers one day. Over the entire timeline, Symantec contains 627,907 nodes (574,733 files and 53,174 machines) and 3,392,983 edges. Its stream elements are collections of edges observed every 10 minutes. We select 30 random starting points from Symantec. Because Symantec is bipartite, it does not contain triangles; so the triangle-based statistics (DegTri, TriCount, and CC) are not evaluated on this network.

---

5 Proprietary data given to us by a collaborator.
4.2 The Necessity of Short Intervals

To determine whether less is sometimes more, we applied belief propagation to different-sized snapshots of file occurrence data that Norton Safewatch users share with Symantec to facilitate reputation-based detection of novel malware files. We structure each snapshot as a bipartite graph between files and machines and apply belief propagation, which spreads the reputation of known malicious and benign files to the machines on which they appear, and then propagates these machine reputation scores back to the preponderance of unlabeled files, as described by Chau et al. [10]. The output is a continuous reputation score for files that is used to determine their probable benignity or maliciousness.

We present our results in Figure 2. We aggregated a day’s worth of file-occurrence data over multiple interval lengths, ranging from 24 1-hour snapshots to a single 24-hour snapshot, and ran belief propagation over each snapshot. Figure 2 reports the percentage of malware samples that were correctly labeled by our solution (true positives) for given percentages of benign files that were incorrectly labeled as malware (false positives) at various threshold values. This figure shows that the value of the data is maximized at shorter time intervals, especially when measured at the small false positive rates at which Symantec’s prevention technologies operate. This happens because of growth in the size of the graph’s connected components as snapshot sizes increase. Networks over shorter time-intervals work much better with Symantec’s data because infected machines receive a short burst of malicious files over a time-span of minutes, while longer snapshots destroy the purity of the graph’s connected components by polluting these bursty malware clusters with increasing numbers of benign files. Effectively, longer snapshots lose the finer granularity needed to detect short-lived trends in the data by increasing the graph’s density.

This sort of density-related problem can occur in other applications as well: as a graph naturally densifies, the nuances that were visible in its sparser form become hidden. Thus, we want snapshots that are short but structured.

4.3 Comparison of Interval Lengths

We now validate the results of ADAGE by showing that the intervals identified by various statistics are suitable for different applications. The amount of structure one requires in a network depends on what one intends to do with that network. For example, finding communities may require more structure than describing the degree distribution. However, directly optimizing for intervals of an application (such as PageRank) may be computationally intensive. Thus, we wish to identify groups of statistics and applications that tend to produce similar results. One can then substitute a cheaper method for a more expensive application. In this way, one would save the cost of running the expensive application on all the sub-intervals that are considered before convergence is achieved.

Recall that data is being observed in terms of stream elements. Each stream element is a collection of edges observed during a fixed time interval (e.g., a day for the Enron Emails dataset or 10 minutes for the Symantec dataset). Thus, rather than report the intervals created by ADAGE as lengths of time, we report them in terms of the numbers of unique edges seen. This allows for an equal
Fig. 3 Heatmap showing Canberra distances between interval length vectors produced by various statistics ('S') and applications ('A'). Lower distances indicate higher similarities. The distinct cluster structure on the lower left indicates that DegDist and DegTri produce similar intervals as PR and Degs. The intervals based on CC are very different than intervals from other statistics and applications.

comparison across different portions of the timeline. Once we have identified the interval lengths produced by ADAGE, we are interested in finding statistics and applications that produce similar intervals.

Given a statistic or application and a set of starting points, ADAGE will produce a set of intervals. Rather than grouping all of these interval lengths together, we directly compare the intervals found by the various statistics and applications at the same starting point. Recall that because data streams have different rates, we measure these interval lengths in terms of number of unique edges, rather than time. For each starting point, we have a set of \( m \) lengths, where \( m \) is the number of statistics or applications applied to that dataset. Thus, if we consider \( n \) starting points, we have \( m \times n \) vectors.

We are interested in determining which statistics or applications find intervals of similar lengths. Such results give us insight into network dynamics. In addition, they allow for the informed use of some cheaply-computed statistic to find appropriate intervals for a more expensively-computed application. To compare the interval lengths produced by statistics \( M_i \) and \( M_j \), we take the normalized Canberra distance between their corresponding vectors.\(^6\) This gives us an \( m \times m \) distance matrix. Figure 3 contains this matrix, in heatmap form, for network Facebook1. We see two clear clusters. On this network, statistics DegTri and DegDist tend to produce intervals of the same length as the applications PR and Degs. These statistics are thus useful if one wishes to run PageRank without incurring the computational cost of using it directly in ADAGE. Figure 4 contains a set of boxplots showing variety in interval lengths on network Facebook1, demonstrating which statistics tend to produce longer or shorter interval lengths.

---

\(^6\) The Canberra distance is defined as \( \sum_{i=1}^{n} \frac{|U_i - V_i|}{|U_i| + |V_i|} \), where \( n \) is the number of dimensions in \( U \) and \( V \). To normalize, we divide by \( n \).
Variety in interval lengths (measured in terms of number of unique edges, a.k.a. interval density) on the Facebook1 network, showing which statistics and applications tend to produce longer or shorter intervals. CC has the longest interval; LCC often has the shortest interval.

To determine which pairs of statistics or applications frequently produce similar interval lengths, we perform the following experiment. For each dataset, for each statistic or application, we have a vector containing interval lengths (measured in number of unique edges). We calculate the normalized Canberra distance between corresponding vectors, which gives us a score between 0 and 1. For each dataset and pair of statistics or applications, we thus get a single distance value, and by aggregating these distance values across all datasets, we can identify pairs of statistics or applications that consistently produce similar interval lengths. Table 2 shows the average and standard deviation for each such pair. We only run belief propagation on the Symantec network, because this was the only dataset that both had positive/negative node labels and was structurally suitable for belief propagation. Recall that we did not run DegTri, TriCount, or CC on the Symantec network since it is bipartite. These cells of the table are thus empty. Additionally, as discussed in the previous section, Eigs was unsuitable for Symantec, Blogs, and IP, and we did not run it on these networks.

Using Table 2, we can identify which statistics and applications consistently produce similar intervals. Here, values closer to 0 indicate that the two statistics or applications are producing similar intervals. We see that DegTri, TriCount, and LCC all produce fairly similar interval lengths; the stability of these triangle-based features occurs at roughly the same time that the size of the giant component stabilizes. The CC statistic generally produces intervals that are very different from those produced by any other statistic or application, and as is illustrated by Figure 4, ADAGE with CC generally converges much later than with any other statistic or application. CC, like DegTri and TriCount, is a triangle-based statistic. However, DegTri and TriCount measure exponents of distributions, and can remain stable even if the number of triangles changes. In contrast, as networks naturally densify, CC will grow, and convergence thus takes longer, repeating the findings in [17]. This example illustrates the importance of selecting an appropriate statistic. In this case, if one were interested in the distribution of triangles, DegTri or TriCount can be used. In contrast, if the absolute number of triangles were of greater importance, CC may be more appropriate.

Interestingly, we see that the PageRank and Degs applications converge at very similar times. This is somewhat unexpected, because although these two applications are both based on the concept of finding ‘central’ nodes, they have very different notions of centrality. Both of these applications give rise to intervals
similar to those found by DegTri. In the next section, we describe how Table 2 can be used as a guide to substitute a cheaply-computed statistic for a more expensive application.

### 4.4 Running Time Analysis

The experiments reported in the previous section help us identify which statistics can be used to approximate intervals for more computationally-intensive statistics or applications. We now present running time characteristics for ADAGE with each statistic and application next.

Our experiments on the Symantec data were run on a 64-bit Linux machine (RedHat Enterprise Linux Server 5.7) with 8 Opteron 2350 quad core processors running at 2.0 GHz, 64GB of RAM, and 100GB disk-quota per user. All other experiments were run on a 64-bit Windows 7 machine with Intel Core i5 quad core processors running at 3.4 Ghz, 32GB of RAM.

Figure 5 lists the average amount of time required to find an interval for each of the statistics or applications. We see that there is a wide range in running time across the various statistics and applications.

Suppose a user is interested in finding the highest PageRank nodes in graph snapshots. Figure 5 shows that ADAGE with PageRank is much more expensive than other statistics or applications, and the cost of finding the optimal intervals for PageRank on very large networks may be prohibitive.

However, Table 2 shows that the DegTri statistic produces intervals similar to those produced by PageRank. Thus, one can use DegTri to find suitable intervals instead of optimizing directly for PageRank.

### 4.5 Comparison to the Most Closely Related Method

We compare ADAGE to the DAPPER algorithm [8] (details in Section 6), which partitions streaming network data based on changes in network structure (rather than network maturity). DAPPER considers a window size and a shift parameter to calculate a frequency change vector. For the window size and shift parameter, values of 3 and 1, respectively, are suggested in [8]. The frequency change vector

Table 2 Average and standard deviation of the normalized Canberra distances between pairs of interval length vectors, across all datasets. Lower values indicate greater similarity. These distances are symmetric, and so the portion under the diagonal is blank. For each application, the closest statistic is bolded. Recall that Belief Propagation was run only on the Symantec dataset, which is bipartite, and thus does not contain any triangles. We thus could not calculate the distances between Belief Propagation and DegTri, TriCount, or CC. We also did not run Eigs on Symantec, and so the distance between Eigs and BP could not be calculated.
Fig. 5 Running time in seconds for ADAGE with each statistic or application. Observe the large variance over the different statistics and application. Some (like DegDist) are very fast, while others (like PageRank) are slow.

Fig. 6 Ratios of interval lengths found by various ADAGE methods to the interval lengths found by DAPPER, where interval lengths are measured in number of unique edges. DAPPER intervals are very different from those found by ADAGE with clustering coefficient, and are similar to those found by ADAGE with Comms.

measures changes in edge persistence from one window to another. If one window has many edges that are not seen in the previous window, then the corresponding unit in the frequency change vector is high. The local maxima of this frequency vector are used in a recursive procedure to partition the timeline.

DAPPER differs from ADAGE in three important ways: (1) it requires the entire data stream at once, (2) it cannot find intervals starting at a specified point, and (3) it cannot be tailored to a specific statistic. We are interested in an online approach that finds one interval beginning at a specified time. The second of these points is especially critical for our experimental set-up: in order to compare intervals across methods, the various methods must each have access to the same data, and thus begin at the same start point. To this end, we modify DAPPER by defining an interval’s endpoint as the second breakpoint found by DAPPER, when given the entire data stream after the specified start time. Note that this modification effectively returns the first complete interval that would be found by the original version of DAPPER, and so we use the same parameters as suggested for the original implementation. In general, this modified version of DAPPER is computationally fast, similar to DegDist. We apply the modified DAPPER to our networks and compare its intervals with those generated by ADAGE. The intervals produced by the modified DAPPER are closest to those produced by ADAGE with the Comms applications, with an average Canberra distance of 0.20 and a standard deviation of 0.09. Our conjecture is that both Comms (which measures the stability of community memberships across timesteps) and DAPPER (which
Fig. 7 Ratios of Jaccard similarities of the top-100 degree centrality nodes from one time step to the next on intervals found by the DegTri statistic, to Jaccard similarities on intervals found by the PR application. Values are generally very close to 1, indicating that DegTri is an appropriate substitute for PR.

identifies changes in the frequency of individual edges) involve specific nodes and edges, rather than general network statistics. DAPPER intervals are least similar to the intervals produced by the CC statistic, with an average Canberra distance of 0.45, with a standard deviation of 0.21.

4.6 Substitution of Intervals

In addition to proposing the ADAGE framework for finding appropriate time intervals, a goal of our work in finding statistics and applications that generate similar intervals is to allow for the substitution of cheaply-computed statistic intervals for more expensive application intervals. To evaluate our success, we perform the following experiment. For applications PR, Comms, and Degs, we find the statistic that produced the most similar intervals according to Table 2. (For BP, we provide a case study in the next section.) Recall that for each of PR, Comms, and Degs, we compared a set found in one time-step to the set found in the next time-step (e.g., we compared community memberships in one aggregated time step to memberships in the next aggregated time step), and sought convergence of these similarity values. We apply this same technique in these experiments.

For each interval \([T_i, T_j]\) found by the closest statistic to some application, we apply the application to intervals \([T_i, T_j]\) and \([T_j, T_j+1]\), and see how similar the results are (i.e., we see how stable the set of highest centrality nodes or community memberships are from one time step to the next, using the same stability measures as before). We then compare these similarity values to the similarity values found when the ADAGE converged with the application itself.

We observe that DegTri produces intervals most similar to those produced by PR. For a DegTri interval, we calculate the top-100 PageRank nodes over that interval. We then add another time step to the interval, and recalculate the top-100 PageRank nodes. We calculate \(v_S\), the Jaccard similarity between these sets. In the same way, we calculate \(v_A\) using the interval found by ADAGE using PR starting at the same start time. We then take the ratio of \(v_S\) to \(v_A\). If this ratio is near one, this confirms the similarity between intervals.

Figure 7 contains these ratios for the DegTri statistic and the PR application for each dataset (except for Symantec, which is bipartite and thus does not contain
triangles). The y-axis of this plot contains these ratios, which we see are very close to 1. We also observed similar results for other applications when comparing with their closest statistic. These results indicate that the intervals produced by these similar statistics are indeed suitable for use with these applications.

4.7 Case Studies

**Belief Propagation (BP).** Consider a scenario in which one wishes to run BP on some network. If there are sufficiently many known labels on the nodes, one could perform cross-validation, directly using ADAGE with BP as the target application. In this case, one would aggregate the data until the true positive rate began to drop. But what would one do in the situation where there are insufficiently many known labels, and so one does not wish to further diminish the training set by performing cross-validation?

We saw in our experiments that ADAGE with the Degree Distribution statistic produced intervals most similar to those found by BP. Thus, one could use ADAGE with Degree Distribution to identify intervals appropriate for running BP.

We apply BP to the Symantec dataset, with the goal of labeling files as malicious or benign. We perform 10-fold cross-validation, and use graphs generated by ADAGE with the Degree Distribution statistic. We calculate the true positive rate at a fixed false positive rate of 0.1%, and compare the results of the ADAGE graphs to graphs formed over 1 hour and 24 hours.

We see that when intervals are identified using ADAGE with Degree Distribution, we get an average true positive rate of 0.878. When aggregated using 1-hour intervals, we get a true positive rate of 0.871, and when aggregated using 24-hour intervals, 0.544. The ADAGE intervals give better results than either of the two fixed-length options. Although one could obtain similar results by using 1-hour intervals, this choice of interval length is unprincipled, and we would not necessarily expect it to work on other datasets.

**Network Similarity.** Our work on ADAGE was motivated by [20], in which the authors evaluated the behavior of a variety of network similarity methods. Their experiments were as follows. One first selects a single ‘reference’ network, and then uses each similarity method (from a collection of methods) to rank various ‘comparison’ networks based on their structural similarity to the reference networks. One then calculates the Kendall-Tau distance between two rankings to determine if two methods are similar. On cross-sectional datasets from a variety of domains (e.g., aggregations of DBLP, LiveJournal, etc), high correlations between similarity methods were observed.

Surprisingly, when applying these similarity methods to daily snapshots of longitudinal datasets (e.g., comparing one daily Twitter replies network to other daily replies networks), these high correlations vanished. Closer examination revealed that the snapshots were unstructured, containing many tiny components. Aggregation of these daily snapshots into multi-day snapshots gave rise to high correlations, but the choice of interval lengths were unprincipled.

ADAGE can be used to identify appropriate time intervals for aggregation of network data for purposes of network similarity. As observed in [20], the lack of network structure was indicated by the presence of many single-edge components.
Fig. 8 Heatmaps showing the Kendall-Tau distances between similarity rankings produced by different methods when Twitter data was aggregated (a) on a daily basis and (b) using ADAGE with the LCC statistic. In (a), distances are high, and thus correlations are low, between the different rankings. This is because the daily networks are not structurally mature. In (b), distances are low, and so correlations are high. This heatmap contains more rows and columns than plot (a) because the daily dataset did not contain enough structure to run several of these network similarity methods (discussed in [20]).

Thus, because the lack of sufficiently large components was the problem, one can choose to use the LCC statistic in ADAGE, which is quickly and cheaply computed. We demonstrate this on a Twitter Replies dataset collected over a period of six months from June 2009. The stream elements are collections of edges observed per day. Over the entire time interval, this network contains 12,941,977 nodes and 44,016,215 edges.

The Twitter data was initially aggregated on a daily basis, which seemed a natural choice. Figure 8(a) shows the Kendall-Tau distances between the rankings produced by the various similarity methods on the daily Twitter data. These snapshots did not even have enough structure for us to run the community based similarity methods. For the other methods, the Kendall-Tau distances are high. This occurs because each network is unstructured, and so the similarity scores are meaningless. Figure 8(b) contains the Kendall-Tau distances between the rankings when the similarity methods were applied to snapshots found by ADAGE, using LCC as the statistic. When using the resulting intervals, we see very low Kendall-Tau distances, similar to what we expected. These intervals range from 5 to 22 days, so daily intervals would clearly have been inappropriate.

5 Discussion

When identifying structurally mature graph snapshots, it is useful to find intervals as short as is practicable. If snapshots are too long, one risks losing a fine-grained understanding of network dynamics. For example, when studying network evolution, it is clearly not useful to aggregate the entire timeline into one single snapshot, even though this snapshot would likely have meaningful structure. In addition, some applications perform better on short, but structurally mature, snapshots than on longer snapshots. We first demonstrated that short intervals can be more valuable than long intervals in Section 4.2 by presenting a study of
belief propagation on the Symantec network. We attempted to label files as benign or malicious, but as the intervals increased and the graph density went up, this task became harder. In addition to this BP experiment, our case study on network similarity (described in Section 4.7) also demonstrated that ADAGE finds proper (i.e., short but structurally mature) intervals.

In Sections 4.3, 4.4, and 4.6, we showed the flexibility of ADAGE by presenting experiments on seven graph statistics, four graph mining applications, and eight real-world graphs from various domains. We described how certain graph statistics with fast running times can effectively be substituted for applications with slower running times. A case in point was the effective substitution of DegTri for PageRank in Section 4.6.

In Section 4.5, we compared ADAGE with its most closely related method: DAPPER [8]. Unlike ADAGE, DAPPER is not an online approach; it cannot discover intervals from a specified time point; and, it is not flexible (in that it cannot take as input various statistics or applications). We modified DAPPER to work as an online approach and compared it to ADAGE. In our experiments, the DAPPER intervals were close to the ADAGE intervals with Comms application. This similarity can be attributed to the fact that both of these approaches – namely, our modified online DAPPER and ADAGE with Comms – track specific nodes and edges rather than general network statistics.

In Section 4.7, we presented two case studies illustrating the use of ADAGE for the tasks of belief propagation and network similarity. In both cases, ADAGE offered a principled way of selecting suitable interval lengths.

6 Related Work

Time-evolving networks are ubiquitous in real-world applications. Previous works either directly address the dynamics of such networks or generalize techniques for static networks to dynamic ones. The most closely related works to ours can be grouped into the following three categories. **Models for time-evolving networks.** A large body of work exists that is concerned with discovering patterns in longitudinal networks. We highlight two of them here. In [17], the authors examine a set of time-evolving networks, and find that they densify. Akoglu et al. [1] propose models for generating time-evolving networks that satisfy patterns such as the eigenvalue power law. Our work proposes a novel framework for leveraging such laws to find structurally mature networks. **Mining time-evolving networks.** Related literature has addressed the problem of designing algorithms to mine different properties of time-evolving networks [3, 4, 21, 22]. In [3, 22], the authors present a framework for analyzing group behavior and finding communities over time. The aim of [4] is to mine frequent patterns of interaction that appear more than expected in a series of snapshots. Sun et al. [21] propose a method for mining patterns and anomalies in large evolving networks. Our work considers such applications when identifying appropriate aggregation intervals. **Aggregation intervals.** The literature on partitioning a data stream is vast. Kiernan et al. [15] identify disjoint ‘summaries’ of an event stream. Keogh et al. [14] present a solution for time series segmentation. Attempts to partition network streams include [8, 16]. In [16], a phone-call network is analyzed using fixed-length intervals. The DAPPER algorithm [8] partitions a timeline of streaming network data
into disjoint interval snapshots by examining edge persistence for change-points. In contrast to these works, we find variable-length intervals for graph data by identifying when a network snapshot is structurally mature.

7 Conclusions

We presented the first extensive study for partitioning a timeline of streaming edge data into variable-length intervals across a variety of network statistics, graph-mining applications, and datasets. We introduced ADAGE, a flexible online framework that can accommodate a variety of network statistics and graph-mining applications to produce structurally mature graph snapshots with convergence guarantees. We observed that for certain real-world applications (e.g., malware detection with belief propagation), graph snapshots aggregated over shorter time intervals can be preferable to snapshots aggregated over longer time intervals. Thus, one should not simply use a long fixed-length interval to generate snapshots. Moreover, we showed that certain pairs of statistics and applications give rise to similar intervals when used in ADAGE. In particular, we observed that the exponent of the mean number of triangles for a given degree (DegTri) is closest to PageRank stability (PR) and degree centrality stability (Degs); and the exponent of degree distribution (DegDist) is closest to community structure stability (Comms) and belief propagation (BP). These similarities are useful because intervals produced by ADAGE with DegTri and DegDist are computationally more efficient than those generated by PR, Degs, Comms, or BP. Lastly, we discussed two use cases of ADAGE: one on malware detection via belief propagation on data from Symantec Inc. and another on network similarity quantification of Twitter replies. As part of future work, we are examining how to incorporate periodicity and other prior knowledge of the network domain into ADAGE.

References

nium: Tera-scale graph mining and inference for malware detection. In: SIAM 
Conference on Data Mining, pp 131–142
11. Dumitras T, Shou D (2011) Toward a standard benchmark for computer se-
curity research: The worldwide intelligence network environment (WINE). In: 
Proceedings of the First Workshop on Building Analysis Datasets and Gath-
ering Experience Returns for Security (BADGERS), pp 89–96
SIGKDD Explorations, vol 5, pp 149–151
time series. In: ICDM, pp 289–296
fects of time window size and placement on the structure of an aggregated 
communication network. EPJ Data Science 1(1):1–16
and shrinking diameters. TKDD 1(1):2
of cascading behavior in large blog graphs. In: SDM, pp 551–556
Spatio-temporal mining of software adoption & penetration. In: ASONAM, 
pp 878–885
work similarity method. In: SDM, pp 1037–1045
free mining of large time-evolving graphs. In: KDD, pp 687–696
munity identification in dynamic social networks. In: KDD, pp 717–726
23. Tsourakakis CE (2008) Fast counting of triangles in large real networks with-
Nature 393(6684):440–442