2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Aging Data in Dynamic Graphs: A Comparative Study

Anita Zakrzewska and David A Bader
School of Computational Science and Engineering
Georgia Institute of Technology, Atlanta, Georgia
Email: {azakrzewska3,bader}@gatech.edu

***Abstract*—Dynamic graphs are used to represent changing relational data. In order to create a dynamic graph representing relationships or interactions over time, it is necessary to choose a method of adding new data and removing, or otherwise de-emphasizing, past data to decrease its influence. In particular, the question of aging edges is new to dynamic graphs and has not been thoroughly studied. In this work, we address the problem of aging vertices and edges to create a dynamic graph from a stream of temporal data. We provide two new methods, *active vertex* and *active edge*, and also evaluate two methods from the literature, *sliding window* and *weight decay*. By analyzing various properties of the dynamic graphs created by each aging method, we provide practitioners with quantitative comparisons. We find several interesting similarities and differences. The *active vertex* and *weight decay* methods reduce the variability over time of several vertex level measures compared to *sliding window* and *active edge*. This means that in practice, *active vertex* or *weight decay* may be more useful if graph stability is preferred, while *sliding window* or *active edge* may be preferred if the graph should be sensitive to changes in the underlying data stream. Each method also differently affects global measures. The most connected graph is produced by *active vertex*, while the most disconnected by *weight decay*. We observe that despite the differences, the graphs produced by each method experience similar types of changes at similar points in time.**

I. Introduction

Graphs are used to represent and analyze a variety of datasets, such as social networks, web traffic, financial transactions, and biological data. Many of these datasets are changing constantly, and to be properly analyzed they must be represented by dynamic graphs.

Whenever dynamic graph analysis is performed, it is necessary to decide how to create a dynamic graph from temporal data. Typically, a dynamic graph is represented by a sequence of graph snapshots, each showing the state of the graph at some particular point in time. To do this, new data must be added and old data removed or de-emphasized in order to model change. The question of how to age off past data is not straightforward. Competing goals include preserving graph stability and detecting new changes.

A. Contributions

While dynamic graph analysis requires a choice about the method of aging edge and vertex data, the analysis of different approaches of doing so remains an open question. This paper addresses the problem of how to age past data when creating a dynamic graph from a stream of temporal data. By aging, we refer to any process that decreases the influence of historical data in a graph. The goals are (1) to provide practitioners with quantitative comparisons of available methods and (2) to provide and compare new alternative approaches.

In section II, we present two new methods, called *active vertex* and *active edge*, and discuss two existing methods from the literature, *sliding window* and *weight decay*. Through experiments on temporal graphs from five social networks in section III, we compare and contrast these approaches to find patterns that consistently appear.

B. Related Work

Previous work has studied how to build a static graph from data. De Choudhury *et al.* [7] examine how to transform raw email communication data into a single, static relationship graph. Structurally different graphs will be formed based on what, if any, communication frequency or volume threshold is set for edge creation. The authors set a threshold on the geometric mean of the annual rate of messages exchanged and study how the graph structure changes as the threshold varies. This work is similar to ours because it addresses the question of how temporal data is used to form a graph. However, the authors only consider creating static graphs, while we address dynamic graph creation.

Clauset and Eagle [5] create a dynamic graph of the Reality Mining dataset using the sliding window approach and examine how the size of the sliding window affects periodicity. They also investigate a "natural" window size based on the power spectra of metrics. The window size in the sliding window approach is also studied by Kossinets and Watts [14], who find that vertex-level properties are less stable than global graph ones. While [5] and [14] only consider a single technique to create dynamic graphs, the goal of our work is to compare different methods.

The work most similar to ours is [16] and [17]. Saganowski *et al.* [17] study the effect on community evolution of using a sliding time window to build a dynamic graph compared to aggregating all past data. The authors examine how the length and amount of overlap of the window size affects how many communities continue, split, merge, dissolve, and appear between consecutive snapshots. Oliveira *et al.* [16] also compare the sliding window approach to aggregating all data. Both works find that removing old data with a sliding window approach causes more change in community structure compared to aggregating all data. Our work differs from these

IEEE/ACM ASONAM 2016, August 18-21, 2016, San Francisco, CA, USA
978-1-5090-2846-7/16/$31.00

in important ways. The previous works only study one method of aging data to create dynamic graphs: the sliding window approach, while we compare two previous and two new methods. To the best of our knowledge, this is the first work that compares the effects of different methods for removing data in dynamic graphs. Second, [16] and [17] each examine only a single dataset. By using multiple datasets, we are able to study which patterns occur consistently for a variety of data sources.

II. Aging Methods

In this section, we describe and discuss methods for creating dynamic graphs from temporal data. While static graphs represent data at one point in time, dynamic graphs represent data over time.

Static graphs are created from a collection of actions or relationships. Deciding how these actions are transformed into edges and vertices of a static graph is not always straightforward. For example, in the case of a graph built from interactions between people, it is necessary to determine under what circumstances such communication should form an edge. One approach is to set a threshold for the frequency of communication over time before an edge is created. Different thresholds will produce very different graphs [7]. When forming a static graph of relationships between people based on common event attendance, various approaches may differently factor how many events two people attended and the number of people at each event [2], again with differing results.

Creating dynamic graphs poses additional complications. In addition to the problems found in the static case, it is necessary to decide when new data is added, when old data is removed, and how the two processes are combined.

As time passes, new actions take place and relationships change. A dynamic graph should reflect this change. In order for a graph to represent the evolving nature of the underlying data, new edges, representing new relationships, may be added, and old edges, representing old relationships, may be down-weighted or removed.

A. Definitions

We make a distinction between the stream of edges that represent actions over time and the graphs built from the stream. Let S be a stream of edges in time, where each $(src, dst, weight, time) \in S$ is a tuple of the two endpoint vertices, the edge weight, and the time of the action represented by the edge. The edge actions occurring during a specific time interval are represented by $S_{l,k} = \{(src, dst, weight, time) \in S \mid l < time \leq k\}$.

Let $G = (V, E)$ be a static graph where V are the vertices of G, and E are the edges. For our notation, each edge $(u, v) \in E$ will have a weight $(u, v).weight$ and a timestamp $(u, v).time$ and each vertex $v \in V$ will have a timestamp $v.time$, corresponding to the latest time of any of its edges. G may be created from data in S using Algorithm 1.

We then represent a dynamic graph $DynG$ by a series of graph snapshots over time. At each point in time, there is a static graph representing the current state. As time moves and the underlying data changes, vertices and edges will be added, removed, and modified to create a new graph snapshot. $DynG = \{G_1, G_2, \ldots, G_n\}$ where $G_t = (V_t, E_t)$ is the state of the dynamic graph at time t. We discuss various methods of creating dynamic graphs from a stream of edges S in the subsequent sections.

Throughout the paper, we will use u as the unit of time and assume that all times are relative to the starting time used. For example, if u is one month, then t will represent the number of months, G_1 is the graph after the first month, and $S_{0,2}$ contains the edges with times within the first two months.

B. Sliding Window

The sliding window approach builds a dynamic graph by creating graph snapshots from intervals, or windows, of a stream of edges. Only the most recent data in the stream, inside the current window, is used to create a graph, while all previous data is forgotten. For example, a daily sliding window would create the most recent graph snapshot using only activity from the last day.

Using the sliding window model, each graph snapshot G_t is created as defined below, where λ is the window size and `ToGraph` is defined in Algorithm 1.

$$G_t = \texttt{ToGraph}(S_{t-\lambda,t})$$

If the unit of time is one week, then $\lambda = 3$ means that the sliding window size is three weeks and the window shifts by one week, causing a $2/3$ overlap between consecutive windows.

Data: $S_{l,k}$
Result: G
initialize empty G;
for $(src, dst, weight, time) \in S_{l,k}$ **do**
  if $(src, dst) \in G$ **then**
    $(src, dst).weight += weight$;
  else
    insert (src, dst) into G;
    $(src, dst).weight = weight$;
  end
  $(src, dst).time = time$;
  $src.time = time$;
  $dst.time = time$;
end

Algorithm 1: `ToGraph`

Consecutive windows can overlap to varying degrees, or not at all, with a higher degree of overlap causing smoother and more gradual changes. The sliding window approach is easily interpreted because it is clear what data each graph snapshot represents. The emphasis on new versus historical data can be adjusted with the window size. However, it does not distinguish well between temporary and persistent relationships. For example, two people who communicate regularly over time, but did not communicate in a given week

will have no edge when using a weekly sliding window. Therefore, the resulting dynamic graph may change greatly between consecutive snapshots. This method is best suited for applications where a graph should represent only the most recent actions, not a balance of historic and new data, and it requires a careful choice of window size and overlap.

The sliding window approach has been used to create dynamic graphs in many works. Non-overlapping windows are used to study the incremental k-clique clustering algorithm [8] and to test a framework for tracking community evolution on the Enron dataset [18]. Overlapping sliding windows have been used to create dynamic graphs to predict vertex centrality [13], to test the DENGRAPH algorithm for incremental community detection [11], and to study community dynamics [10] [9]. In [14], a sliding window of 60 days is used to create graphs of email exchanges. Unlike many other sliding window approaches, instead of summing the number of edges to obtain a weight, the weight is set to the geometric rate of bilateral email exchange during the 60 day window. Clauset and Eagle [5] analyse a network of physical contact from the Reality Mining study and investigate the effect of the size of the window.

C. Edge Weight Decay

The second method discussed in this work is the edge weight decay approach in which, over time, the weight of old edges is decreased. The philosophy behind this is to treat new data as more important than old data, while also allowing stronger (higher weight) relationships to last longer than weaker (lower weight) ones. This is done by creating a new graph snapshot from a weighted combination of the previous graph and the new edges. Edges below a weight threshold ϵ are removed to eliminate relationships that are no longer relevant. Throughout the text we refer to this method as *weight decay*. Each graph snapshot created by *weight decay* is defined as:

$$G_t = \texttt{Decay}(G_{t-1}, \alpha) + \beta * \texttt{ToGraph}(S_{t-1,t})$$

where `Decay` is defined in Algorithm 2.

Data: $G = (E, V)$, α
Result: G
for $(src, dst) \in E$ **do**
  $(src, dst).weight = (src, dst).weight * \alpha$;
  if $(src, dst).weight < \epsilon$ **then**
    remove (src, dst) from G;
  end
end

Algorithm 2: `Decay`

In the edge weight decay method, edges that represent frequent communication or strong relationships will last for a longer time, which may create a dynamic graph with smoother transitions between snapshots compared to the sliding window approach. The parameters α and β control the relative importance of historic and new data, while ϵ represents the lowest weight edge that will remain in the graph. Compared to the sliding window approach, the interpretation of a graph snapshot based on the parameters is more difficult because there is no simple interpretation, such as "activity from the past week".

Variations of the edge weight decay method have been used in the literature to create dynamic graphs. [4] predicts future links in a co-authorship network by exponentially decaying weights, to strike a balance between the the recency of a co-published paper and the number of co-published papers. The authors in [6] summarize past behavior by using a linear combination of new and historical data and keeping only the top k edges for each vertex. This same method is used in [12] to predict future data. In [3], the authors predict future edges in a mobile network using exponential aging.

D. Active Graph

In addition to *sliding window* and *weight decay* from the literature, we also propose a new approach to creating a dynamic graph from a stream of edges. This approach relies on identifying active parts of the graph and preserving data in these active parts. The idea behind our new approach stems from the fact that entities join and leave social networks and the amount of time a given entity or relationship is active in the network varies greatly. For example, many users join an online social network and soon become inactive [19] [15]. The edges representing their initial activity upon joining can likely be removed quickly because they no longer participate. Other users, however, remain active in the social network. Their information may be more important and therefore kept in the dynamic graph for longer.

We call the two variations the *active vertex* and *active edge* methods. *Active vertex* keeps track of the last time each vertex was active (had a new edge in the stream S). Vertices whose last activity is within a specified window of time, τ_V, are considered active and we keep all edges whose two endpoint vertices are active. All other edges are removed. Each graph snapshot using *active vertex* is defined as:

$$G_t = \texttt{CheckActiveVertices}(G_{t-1} + \texttt{ToGraph}(S_{t-1,t}), t)$$

where `CheckActiveVertices` is defined in Algorithm 3.

For the second variation, *active edge*, we keep track of the last time an edge's weight was incremented (the last time an activity occurred between the two vertices). Edges whose last activity is within a specified window, τ_E, are considered active and their weights do not decrease. Instead, inactive edges are removed from the graph regardless of their weights. Each graph snapshot using *active edge* is defined as:

$$G_t = \texttt{CheckActiveEdges}(G_{t-1} + \texttt{ToGraph}(S_{t-1,t}), t)$$

where `CheckActiveEdges` is defined in Algorithm 4.

The *active vertex* and *active edge* methods can be useful in representing the history of activity in active parts of the graph, while forgetting the inactive portions. They may also be useful in storing important parts of the graph when memory size is

Data: $G = (E, V)$,t
Result: G
for $(src, dst) \in E$ **do**
  if $t - src.time > \tau_V$ *or* $t - dst.time > \tau_V$ **then**
    remove (src, dst) from G;
  end
end

Algorithm 3: `CheckActiveVertices`

Data: $G = (E, V)$,t
Result: G
for $(src, dst) \in E$ **do**
  if $t - (src, dst).time > \tau_E$ **then**
    remove (src, dst) from G;
  end
end

Algorithm 4: `CheckActiveEdges`

limited because edges are either preserved with cumulative weights, or removed completely. Similarly, they may be useful for visualization to represent accumulated relationships only of active vertices, thus reducing visual clutter.

E. Setting Parameters for Each Aging Method

In section III, we compare the four methods described. To make a fair comparison, we will set the various parameters to make all methods as similar as possible. The unit of time u is the same for each method so that the same number of edges from S is added for each new snapshot. To make *weight decay* similar to *sliding window*, we set $\alpha = \epsilon^{\frac{1}{\lambda}}$. An edge with weight 1 will then exist in the graph for λ snapshots, just as in *sliding window*. Higher weighted edges will stay in the graph for longer. $\beta = 1$ so that new data has the same weight as in the other methods.

For *sliding window*, only data within λ time of the current time is included in a graph snapshot. For *active vertex*, we can set the threshold of a vertex being active τ_V to equal λ from *sliding window*. Similarly, for *active edge*, τ_E is set equal to λ. Note that setting $\tau_V = \lambda$ means that the set of vertices with degree greater than zero of a graph snapshot G_t produced by *sliding window* and by *active vertex* will be the same. Similarly, setting $\tau_E = \lambda$ causes the set of edges in a graph produced by *sliding window* and by *active edge* to be the same (the edge weights may differ though).

III. Results

A. Experimental Setup

We compare the four methods for creating dynamic graphs using the five social network datasets listed in Table I. These are graphs of co-authorship relationships from Dblp, emails from the Enron dataset, thread replies on the Slashdot website, YouTube friendships, and Facebook wall posts, each from the Konect website [1]. For a proper comparison, we set the parameters of each method as described in section II-E. The unit of time, or the amount of new data added at each time step, is the same for each method and given in Table I. For each dataset apart from Dblp, we removed beginning and ending times with few edges and then set u so that when $\lambda = 3$, approximately four non-overlapping windows fit into the time range. For the Dblp dataset, we used 30 years of data from 1984 to 2014 and chose a window size of 3 years based on the assumption that research relationships may change significantly over this time interval. For *sliding window*, we set $\lambda = 3$, yielding a $2/3$ overlap between consecutive windows. Parameters of other methods are then set as described in section II-E. Both τ_V and τ_E are set to equal λ. ϵ should be a small value so we chose 0.1 and therefore $\alpha = 0.1^{\frac{1}{3}} \approx 0.464$.

TABLE I: Datasets and parameters used.

Graph	Vertices	Edges	Time Unit u	Window Size
DBLP	1,314,050	18,986,618	1 year	3 years
Enron	87,273	1,148,072	4 months	1 year
Slashdot	51,083	140,778	1 month	3 months
YouTube	3,223,589	9,375,374	2/3 months	2 months
Facebook	46,952	876,993	4 months	1 year

The above parameters are used for results in Figures 1, 2, and 3. In the experiments shown in Figure 4, we vary the amount of overlap between consecutive windows of *sliding window*, while keeping the number of months in each sliding window the same as shown in Table I. The details are further explained in section III-D.

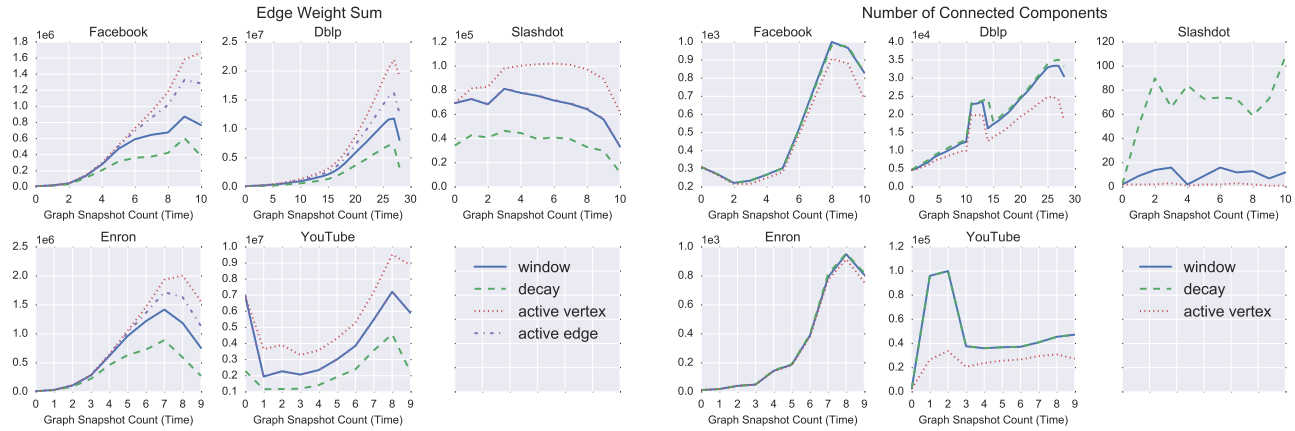
B. Global Properties

Based on the chosen parameters, *sliding window* and *active vertex* will result in graphs with the same number of vertices, while *sliding window* and *active edge* will produce graphs with both the same number of vertices and edges. *Weight decay* will output graphs with at least as many vertices and edges as *sliding window*.

In Figure 1a, the sum of edge weights of the dynamic graphs is shown over time. For all graphs, *active vertex* produces the highest edge weight sum. Interestingly, *weight decay*, although it stores more edges than *sliding window* and *active edge*, produces the lowest edge weight sum. This suggests that edge weights in the graphs produced by these datasets are generally low and fall below the removal threshold ϵ quickly.

The effect of *active vertex* and *weight decay* is the opposite on graph connectivity. Figure 1b shows that *weight decay* produces graphs with a larger number of connected components, meaning that it creates a more disconnected graph. *Active vertex*, on the other hand, produces the fewest connected components, meaning that the graph is more connected.

It is interesting to see that while the scores produced by different methods differ, they follow the same pattern by increasing and decreasing at the same time. This suggests that when the method parameters are matched using the method described in section II-E, similar types of changes can be detected by any of the four methods. For example, the points in time at which a graph disconnects will be approximately the



(a) The sum of edge weights over time.

(b) The number of connected components over time.

Fig. 1: Global properties of the dynamic graphs are shown over time. The x-axis shows the graph snapshot count.

same for each method, marked by an increase in the number of connected components.

C. Vertex Properties

Figure 2 shows results for several vertex-level centrality scores over time. The vertex centrality properties analyzed are unweighted degree (number of edges for a vertex), weighted degree (sum of edge weights of a vertex), and betweenness centrality (the number of shortest paths between all vertices that pass through the given vertex). For each of the three properties, we compute the Spearman's rank correlation between consecutive snapshots. This measures how much the ranking of vertices according to a particular property (such as degree) changes from one snapshot to the next. We also compute the overlap, using the Jaccard index, between the top 1% of vertices (according to degree or weighted degree or betweenness centrality) of consecutive snapshots. Both the overlap and the rank correlation indicate how stable over time the properties measured are. The x-axis shows the graph snapshot count. Note that for statistics using unweighted edges, results for the active edge method are the same as for the sliding window. For rank correlations, vertices that appear in either of two consecutive snapshots are used.

As with the global properties, the vertex-level properties produced by each method increase and decrease at the same time. The graphs produced by each method change in similar ways at similar points in time, even though the magnitude of change may be different.

For all measures, the score rank correlation coefficients and top 1% overlap are highest for *active vertex*. This means that *active vertex* reduces the variability of vertex centrality over time. It produces graphs that are less sensitive to quick changes in time. This can be useful for applications where the graph should change gradually over time. *Weight decay* also has a relatively high rank correlation scores and high overlaps of the top 1% of vertices for unweighted degree and betweenness centrality. On the other hand, it produces the most variation in weighted degree centrality.

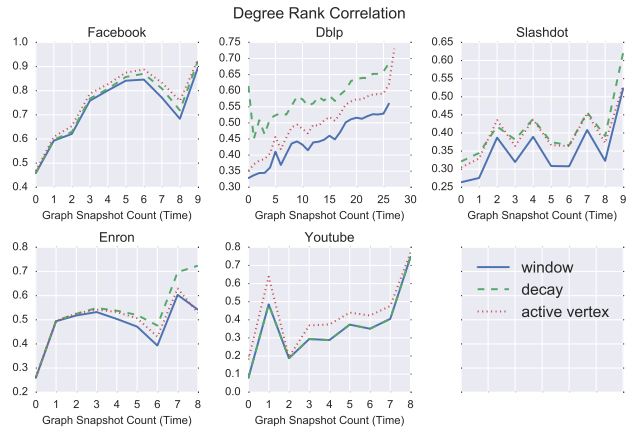
On the other hand, *sliding window* has the lowest rank correlation scores and the lowest overlap of the top 1% of vertices. This means that *sliding window* produces dynamic graphs with a high variability of vertex centrality over time. High degree vertices will more easily become low degree vertices and vice versa. Because the graphs produced by *sliding window* may change quickly, it may be less suitable for applications that require gradual evolution. On the other hand, it may be more appropriate for creating graphs that are sensitive to recent changes.

The fact that *active vertex* and *weight decay* reduce the variability of vertex centrality compared to *sliding window* is even more pronounced when we consider only medium and high degree vertices instead of all vertices. Figure 3 compares the betweenness centrality rank correlation of all vertices, as in Figure 2e, to the rank correlation of only vertices with degree greater than 9 (a vertex must have such a degree in at least one of the two consecutive snapshots). In the top plots, each bar shows the ratio of the average betweenness centrality rank correlation of *active vertex* compared to the average rank correlation of *sliding window*. Values above 1 indicate that *active vertex* produces higher rank correlations than *sliding window*. The bottom plots show the same ratio for *weight decay* compared to *sliding window*.

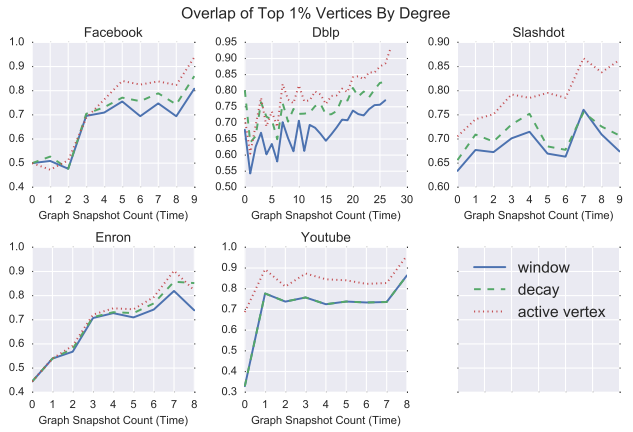
Plots on the left show rank correlations using all vertices and those on the right use only vertices with degree greater than 9 in either consecutive snapshot. It is clear that ratios are higher when the rank correlation uses only the vertices of degree greater than 9. This means that the betweenness centrality variability of medium and high degree vertices is more affected by the choice of aging method than is the variability of low degree vertices.

D. Effect Of Overlap

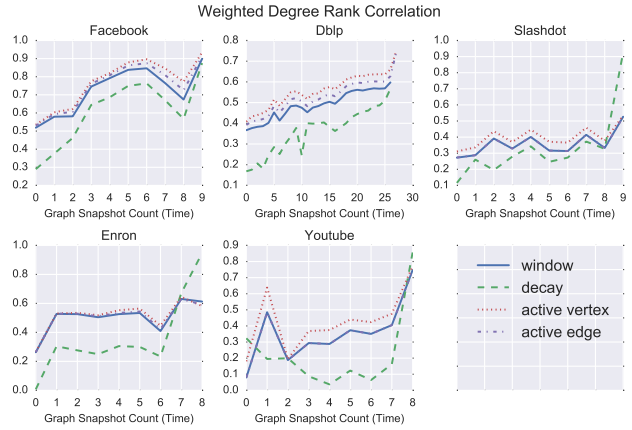
The experiments for Figures 2 and 3 used parameters based on consecutive snapshots of *sliding window* overlapping by



(a) The Spearman's rank correlation of unweighted degrees is shown over time for each method.

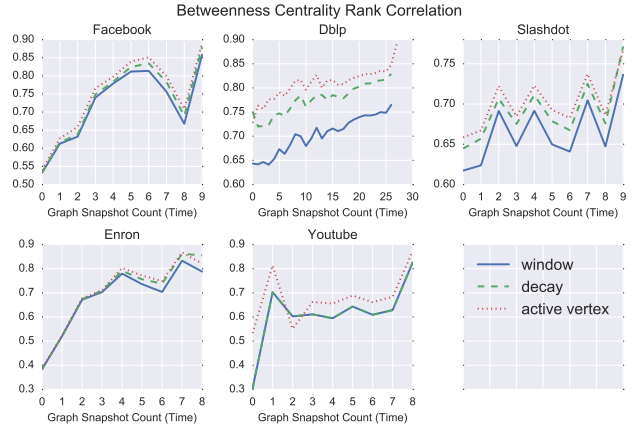


(b) The overlap (Jaccard index) between the top 1% of vertices based on unweighted degree from consecutive snapshots is shown.

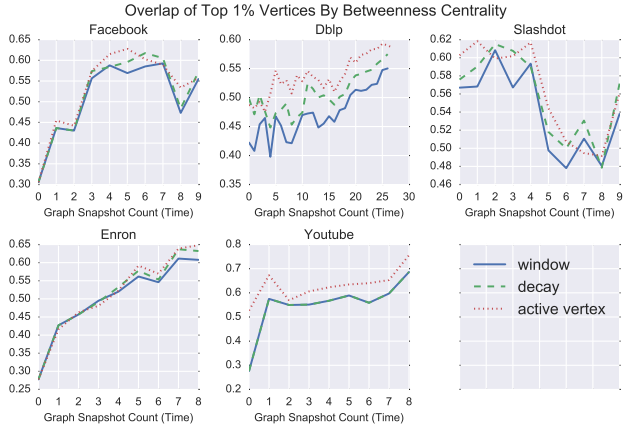


(c) The Spearman's rank correlation of weighted degrees is shown over time for each method.

(d) The overlap (Jaccard index) between the top 1% of vertices based on weighted degree from consecutive snapshots is shown.



(e) The Spearman's rank correlation of betweenness centrality is shown over time for each method.



(f) The overlap (Jaccard index) between the top 1% of vertices based on betweenness centrality from consecutive snapshots is shown.

Fig. 2: Local, vertex-level properties of the dynamic graphs produced by each method are shown over time. The x-axis shows the graph snapshot count. Note that for statistics using unweighted edges, results for *active edge* are the same as for *sliding window*. For rank correlations, vertices that appear in either of two consecutive snapshots are used.

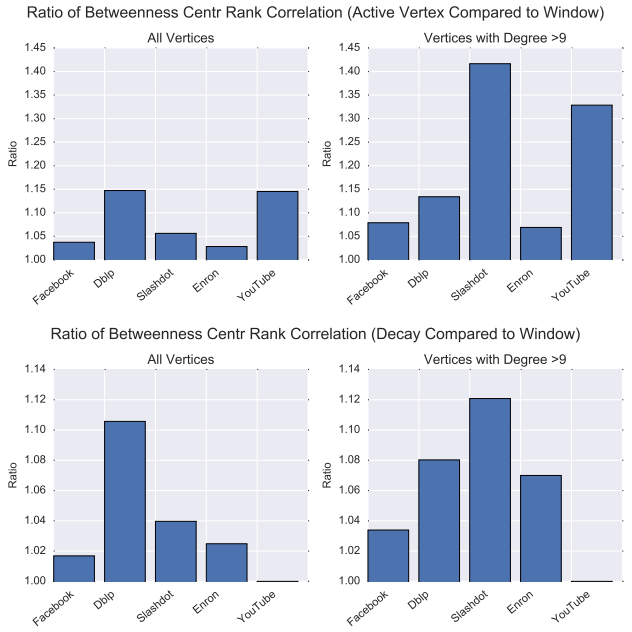


Fig. 3: The ratio of the average betweenness centrality rank correlation of *active vertex* (top) and *weight decay* (bottom) compared to that of *sliding window* is shown. Plots on the left show rank correlations using all vertices an those on the right use only vertices with degree greater than 9 in either consecutive snapshot.

$2/3$, as described in section III-A. In Figure 4, we show how the average scores of different methods change as the amount of overlap between consecutive snapshots increases and decreases. For *sliding window*, we test overlaps of 0, $1/2$, $2/3$, $3/4$, and $4/5$, while keeping the number of months in each window constant. An overlap of $\frac{h-1}{h}$ requires $\lambda = h$ and because the number of months in each windows equals $u\lambda$, the unit of time u must decrease as λ increases.

The parameters of the other methods are set to match as described in section II-E. For *active vertex* and *active edge*, τ_V and τ_E are set equal to λ. For *weight decay*, $\epsilon = 0.1$ and values of $\alpha = \epsilon^{\frac{1}{\lambda}}$ are then 0.0999, 0.316, 0.464 0.562, and 0.630 for different overlaps. The unit of time u is the same for each method.

In Figure 4, the overlap increases left to right on each x-axis. The difference between *weight decay* scores and *sliding window* scores increases as the amount of overlap decreases. This means that when a large amount of data is added and removed between graph snapshots, *weight decay* reduces the variability over time of vertex centrality even more compared to *sliding window*. The same holds to a lesser degree for *active vertex*. The lower the degree of overlap between consecutive snapshots, the greater the difference between methods.

IV. Conclusion

This paper addresses the question of aging data for the creation of dynamic graphs. In addition to existing methods, we provide a new approach based on the concept of active vertices and edges. By analyzing several global and vertex-level graph properties, we find the differences and similarities between dynamic graphs created by each aging approach.

The *active vertex* and *weight decay* methods both reduce the variability of vertex centrality scores over time, especially for medium and high degree vertices, compared to *sliding window* and *active edge*. This difference is greater when more changes are accumulated between consecutive graph snapshots. In practice, *active vertex* or *weight decay* may be more useful if graph stability is preferred, while *sliding window* or *active edge* may be chosen if a faster reflection of changes in the underlying data is needed. The choice of a method matters more when more data is added between snapshots. However, *active vertex* and *weight decay* have opposite effects on graph connectivity. *Active vertex* method decreases the number of connected components, while *weight decay* increases it. This suggest that the types of edges that are kept in the graph by the two methods are different.

Despite these differences, we find that if each method's parameters are carefully chosen, they all produce dynamic graphs with very similar patterns. The dynamic graphs produced by each method experience similar types of changes at approximately the same time. This is an important consideration for applications where the goal is to monitor the graph and detect change-points in time.

The effects on the graph properties studied here will help practitioners understand the consequences of choosing a particular method of removing old data. Because the topic of aging data in dynamic graphs was previously largely unexplored, we focus on basic graph properties. Future work will study the effect on more complex graph measures, such as community evolution. We expect that the method chosen will have a strong effect both on the communities found and how much they change between consecutive snapshots.

Acknowledgments

This work was partially sponsored by Defense Advanced Research Projects Agency (DARPA) under agreement #HR0011-13-2-0001 (DARPA PERFECT). The content, views and conclusions presented in this document do not necessarily reflect the position or the policy of DARPA or the U.S. Government, no official endorsement should be inferred.

References

[1] The koblenz network collection KONECT, 2016.
[2] Stephen P Borgatti and Daniel S Halgin. Analyzing affiliation networks. *The Sage handbook of social network analysis*, pages 417–433, 2011.
[3] Shu-Yan Chan, Pan Hui, and Kuang Xu. Community detection of time-varying mobile social networks. In *Complex Sciences*, pages 1154–1159. Springer, 2009.
[4] Hung-Hsuan Chen, Liang Gou, Xiaolong Luke Zhang, and C Lee Giles. Predicting recent links in foaf networks. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 156–163. Springer, 2012.
[5] Aaron Clauset and Nathan Eagle. Persistence and periodicity in a dynamic proximity network. *arXiv preprint arXiv:1211.7343*, 2012.
[6] Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Computational methods for dynamic graphs. *Journal of Computational and Graphical Statistics*, 2012.

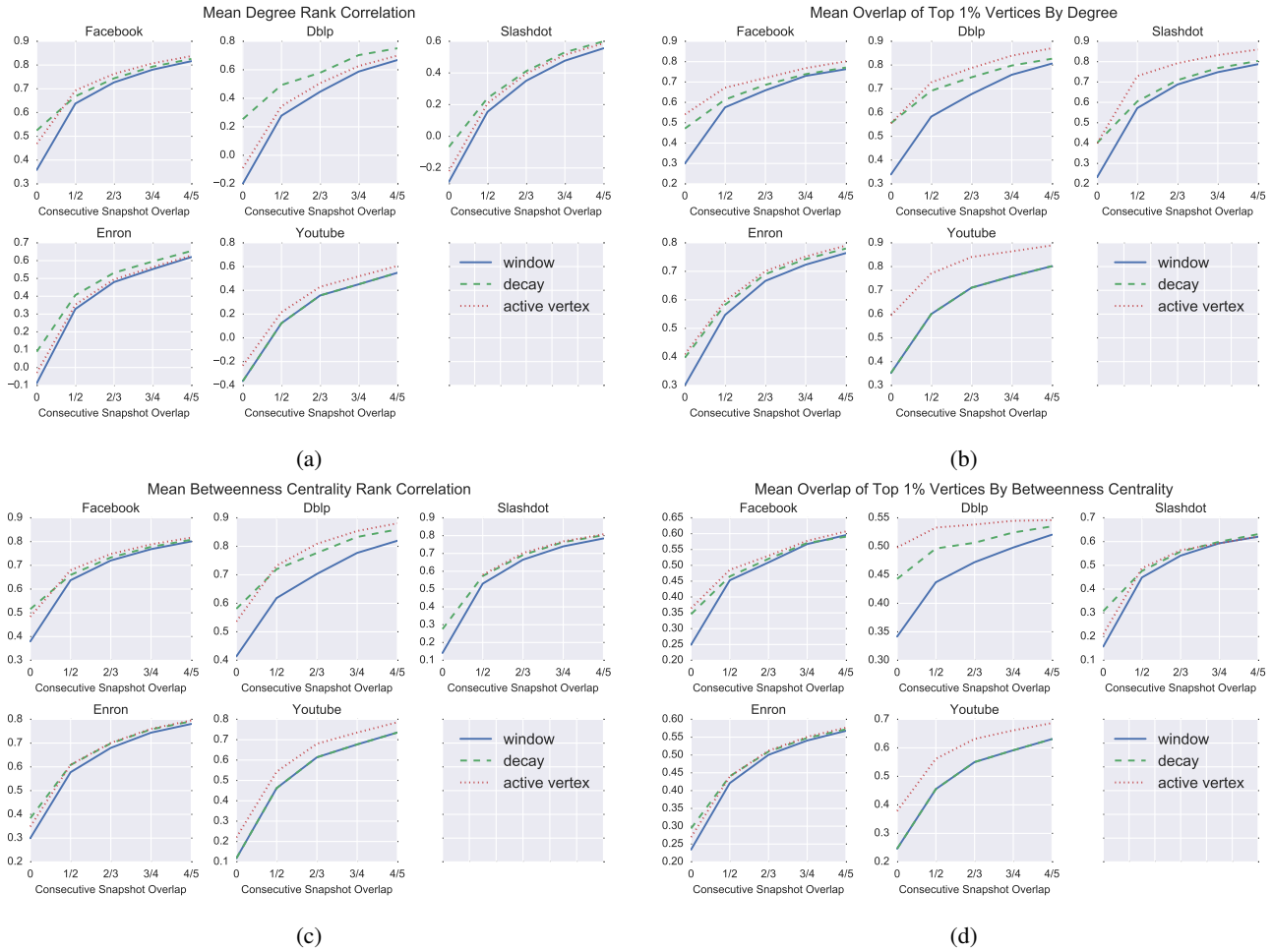


Fig. 4: Plots show how the average score changes as the amount of overlap between consecutive snapshots changes. The x-axis shows the overlap between consecutive snapshots produced by the sliding window method. Parameters of other methods are set to match this. The y-axis shows the average score over all snapshots for that amount of overlap.

[7] Munmun De Choudhury, Winter A Mason, Jake M Hofman, and Duncan J Watts. Inferring relevant social networks from interpersonal communication. In *Proceedings of the 19th international conference on World wide web*, pages 301–310. ACM, 2010.

[8] Dongsheng Duan, Yuhua Li, Ruixuan Li, and Zhengding Lu. Incremental k-clique clustering in dynamic social networks. *Artificial Intelligence Review*, 38(2):129–147, 2012.

[9] Tanja Falkowski, Jörg Bartelheimer, and Myra Spiliopoulou. Community dynamics mining. In *ECIS*, pages 318–329, 2006.

[10] Tanja Falkowski, Jorg Bartelheimer, and Myra Spiliopoulou. Mining and visualizing the evolution of subgroups in social networks. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 52–58, 2006.

[11] Tanja Falkowski, Anja Barth, and Myra Spiliopoulou. Studying community dynamics with an incremental graph mining algorithm. *Proceedings of the 14th Americas Conference on Information Systems (AMCIS)*, pages 1–11, 2008.

[12] Shawndra Hill, Deepak K Agarwal, Robert Bell, and Chris Volinsky. Building an effective representation for dynamic networks. *Journal of Computational and Graphical Statistics*, 2012.

[13] Hyoungshick Kim, John Tang, Ross Anderson, and Cecilia Mascolo. Centrality prediction in dynamic human contact networks. *Computer Networks*, 56(3):983–996, 2012.

[14] Gueorgi Kossinets and Duncan J Watts. Empirical analysis of an evolving social network. *science*, 311(5757):88–90, 2006.

[15] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–470. ACM, 2008.

[16] Márcia Oliveira, Américo Guerreiro, and João Gama. Dynamic communities in evolving customer networks: an analysis using landmark and sliding windows. *Social Network Analysis and Mining*, 4(1):1–19, 2014.

[17] Stanislaw Saganowski, Piotr Bródka, and Przemyslaw Kazienko. Influence of the dynamic social network timeframe type and size on the group evolution discovery. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 679–683. IEEE, 2012.

[18] Mansoureh Takaffoli, Farzad Sangi, Justin Fagnan, and Osmar R Zaïane. A framework for analyzing dynamic social networks. *Applications of Social network Analysis (ASNA)*, 2010.

[19] Xiaohan Zhao, Alessandra Sala, Christo Wilson, Xiao Wang, Sabrina Gaito, Haitao Zheng, and Ben Y Zhao. Multi-scale dynamics in a massive online social network. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 171–184. ACM, 2012.