Ranking in Dynamic Graphs Using Exponential Centrality

Eisha Nathan[1(✉)], James Fairbanks[2], and David Bader[1]

[1] School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, USA
{enathan3,bader}@gatech.edu
[2] Georgia Tech Research Institute, Atlanta, USA
james.fairbanks@gtri.gatech.edu

Abstract. Many large datasets from several fields of research such as biology or society can be represented as graphs. Additionally in many real applications, data is constantly being produced, leading to the notion of dynamic graphs. A heavily studied problem is identification of the most important vertices in a graph. This can be done using centrality measures, where a centrality metric computes a numerical value for each vertex in the graph. In this work we focus on centrality scores obtained from the computation of the matrix exponential. Specifically, we present a new dynamic algorithm for updating exponential centrality-based values of vertices in evolving graphs. We show that our method is faster than pure static recomputation, obtaining about 16× speedup in real-world networks while maintaining a high quality of recall of the top ranked vertices in graphs. Moreover, we do not see a deterioration of the quality of our algorithm over time as more data is inserted into the graph.

1 Introduction

Network analysis has become an increasingly important research area with applications to biological, societal, or financial data [9,24]. One of the most fundamental questions arising from the analysis of complex networks is to determine the "most important" vertices in a graph. Vertex importance is termed as centrality, where a centrality metric typically provides a numerical value for each vertex in the graph [6,19]. Centrality scores can then be turned into rankings on the vertices of a graph [16,20], where a higher centrality value indicates a more important (or highly ranked) vertex. For many application purposes, it is primarily the highly ranked vertices that are of interest [3]. Consider the results of a Google search, where a user desires the most relevant results to the original query to appear first. Additionally, in a network modeling disease spread, an analyst might be interested in identifying the sites that contribute to the formation of epidemics. These queries are answered by identifying the highly ranked vertices in the respective graphs. In this paper, we focus on the centrality values derived from the matrix exponential.

© Springer International Publishing AG 2018
C. Cherifi et al. (eds.), *Complex Networks & Their Applications VI*,
Studies in Computational Intelligence 689, https://doi.org/10.1007/978-3-319-72150-7_31

Additionally, large real world data sets today are constantly evolving, and the analysis of changing relationships in networks is an important aspect of modern data analysis. These changing relationships in data over time can be represented by a dynamic graph. Our work extends the study of centrality metrics on static graphs to the study of centrality on dynamic graphs. Specifically, we want analytics that can update quickly as the underlying graph changes as well. Any algorithm to compute a centrality metric can be used to update the centrality metric in dynamic graphs by recomputing the centrality metric from scratch every time the graph changes. We term this naive approach as *static recomputation*. However, this becomes very costly as the number of changes to the graph increases. For low latency applications such as cyber-network monitoring, financial fraud detection, or social media applications it is important to update centrality values in a dynamic graph efficiently to avoid a full static recomputation.

This work develops a new dynamic algorithm for updating exponential-based centrality scores in evolving graphs. Our method is faster than standard static recomputation and maintains high recall of the highly ranked vertices over time. We test our method on both synthetic and real-world dynamic graphs. The remainder of the paper is organized as follows: Sect. 2 gives some background and definitions required to understand the problem. We outline related work in the literature in Sect. 3. Section 4 presents our new dynamic algorithm and Sect. 5 contains experimental results on both synthetic and real-world graphs. In Sect. 6 we conclude.

2 Background and Definitions

Let $G = (V, E)$ be a graph where V is the set of n vertices and E the set of m edges. One very popular representation of a graph is the use of an adjacency matrix A, which is an $n \times n$ matrix of 1 s and 0 s where $A(i, j) = 1$ if $(i, j) \in E$, 0 otherwise. In this work we deal with undirected, unweighted graphs so $\forall(i, j)$, $A(i, j) = A(j, i)$ and all edge weights are 1; however, our results generalize to weighted and directed networks where applicable. To represent a dynamic graph, we take snapshots of the current graph at different points in time. Let $G_t = (V_t, E_t)$ and A_t be the snapshot of graph G and its corresponding adjacency matrix at time t. Here, we assume a fixed vertex set so $\forall t, V_t = V$, but edges are allowed to change over time and E_t denotes the edge set at time t. Let ΔA be the matrix denoting the edges that are being inserted into the graph at time $t+1$, or intuitively the change between the graphs at time t and $t+1$. For example, if we insert an edge between vertices v and u at time t, then $\Delta A[v, u] = 1$ at time t. Given the previous adjacency matrix at time t, we can write the new adjacency matrix at time $t + 1$ as $A_{t+1} = A_t + \Delta A$.

In [11], the authors introduced *subgraph centrality* as a measure of calculating vertex importance. Subgraph centrality is determined by the diagonal elements of some matrix function applied to the adjacency matrix A of the graph under study. A frequent function of choice is the matrix exponential e^A [10]. Consider

the power series expansion of e^A [14]:

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots + \frac{A^k}{k!} + \cdots = \sum_{k=0}^{\infty} \frac{A^k}{k!},$$

where I is the $n \times n$ identity matrix. It is a well-known fact from graph theory that $A^k(i,j)$ counts the number of walks of length k between vertices i and j, where a walk of length k in a graph is a sequence of vertices $v_1, v_2, \cdots, v_k$ and $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k-1$. Therefore, the diagonal elements of e^A, $e^A(i,i)$, count the number of closed walks (starting and ending at the same vertex) centered at vertex i weighting a walk of length k by $\frac{1}{k!}$. In this paper, we use the diagonal elements of the matrix exponential, $e^A(i,i)$ as the centrality scores for the vertices. An alternate means of calculating centrality scores from the matrix exponential is to use the row sums of e^A, since in practice this is faster than obtaining the diagonal elements, which requires computation of the entire matrix. However, various results in previous literature have shown that these two methods (row sums versus only the diagonal elements) often produce fairly different rankings and so we cannot simply replace one with the other [5]. Since our analysis of exponential centrality requires calculating the entire matrix, which is a dense matrix, this work focuses on medium sized graphs; however future work can consist of using methods to approximate the matrix exponential to scale to larger graphs.

3 Related Work

Several centrality measures can be expressed as a function of the adjacency matrix of a graph. PageRank is a common method for ranking vertices in graphs, where a high score means random walks through the graph tend to visit the highly ranked vertices, and was first introduced rank webpages in a web search [25]. The solution $\mathbf{x}$ to the equation $(I - \alpha A^T D^{-1})\mathbf{x} = (1-\alpha)\mathbf{v}$ gives the desired PageRank vector, where $\mathbf{v}$ is usually drawn from a uniform distribution, α is typically set to 0.85 [12], and D is the matrix of diagonal elements from A. Eigenvector centrality is another linear algebra based centrality measure for weighing relative importance of vertices in networks, by examining the eigenvector corresponding to the largest eigenvalue of the adjacency matrix [7]. Eigenvector centrality takes into account both direct connections to vertices as well as indirect, thereby taking into consideration all walks through the network. It is defined as the solution $\mathbf{x}$ to the equation: $A\mathbf{x} = \lambda\mathbf{x}$, where λ is the largest eigenvalue of A. In this work we focus on the centrality scores obtained by using the diagonal elements of the matrix exponential e^A as discussed in Sect. 2. Many previous works in the literature have extensively studied the matrix exponential, but all of these have been with respect to static (non-evolving) graphs. For example, [9] uses row sums to calculate vertex importance, [4] applies calculations involved in computing the matrix exponential to the identification of hubs in directed networks, and [14] presents methods to approximate the matrix exponential.

Although the study of centrality measures in static networks has been used for a variety of applications [23], several networks are constantly changing and it is therefore important to develop definitions of centrality measures for dynamic graphs. One such work considers a temporal network as a sequence of layers and examines centrality scores between these layers [8]. The most popular approach when analyzing temporal networks considers path-based centrality in static graphs and extends the metric using *time-respecting paths*, where a time-respecting path as one that allows up to an unlimited number of edge traversals during a particular time step. Several works have examined extending PageRank for dynamic networks. One such example is seen in [18], where the authors update the eigenvalue formulation of PageRank to update the vector using the power method. By using exact [21] and approximate [27] aggregation techniques, they efficiently update the transition matrix after updates to the graph are made. Temporally extended versions of betweeness centrality [2], communicability [13], and closeness centrality [26] have also been studied in the past literature. For a comprehensive survey of dynamic centrality measures, see [28]. However, as far as the authors are aware there has been no prior work in developing an algorithm to update the matrix exponential for dynamic graphs.

4 Methodology

The goal of our dynamic algorithm is to prune unnecessary computation when calculating the updated centrality scores of the vertices in the graph after edge updates occur. Therefore, our algorithm uses the computations from the previous timestep in the calculation of the scores in the current timestep. This forms the basis of our dynamic algorithm. We obtain updated snapshots of the adjacency matrix at time $t+1$ as $A_{t+1} = A_t + \Delta A$, where ΔA represents the edge udpates occurring at time $t+1$.

The end goal is to calculate $e^{A_{t+1}}$, or equivalently $e^{A_t+\Delta A}$. Since we are working with exponentials, a naive first pass algorithm is to attempt to exploit basic properties of exponentials, namely the additive property. However, the additive property of exponentials fails for matrices unless we have commutativity: for $n \times n$ matrices $A = B + C$, $e^A \neq e^B + e^C$ unless $BC = BC$. Since we cannot trust graph updates to be commutative, this naive additive property alone is not sufficient for our purposes and we cannot simply compute $e^{A_{t+1}}$ as $e^{A_t} + e^{\Delta A}$. However, there is still a relationship between the parts of the sum for the matrix exponential as stated in Theorem 1 that we can use to develop a streaming algorithm for the matrix exponential in dynamic graphs.

Theorem 1. *Suppose $A = B + C$ where A, B, and C are $n \times n$ matrices. Then the exponential of A is related to the exponentials of B and C by the Trotter product formula [29]:*

$$e^A = \lim_{m \to \infty} (e^{B/m} e^{C/m})^m.$$

Furthermore, the Trotter result can be used to approximate e^A by using the approximation [22]:

$$e^A \approx (e^{B/m} e^{C/m})^m.$$

Suppose we have the matrix exponential of A at time t, e^{A_t}. Given edge updates ΔA to the graph, our goal is to compute the updated matrix exponential $e^{A_{t+1}}$ with minimal computation. Using Theorem 1, we can calculate $e^{A_{t+1}}$ as:

$$\begin{aligned} e^{A_{t+1}} &= e^{A_t + \Delta A} \\ &\approx (e^{A_t/m} e^{\Delta A/m})^m \end{aligned}$$

As the value of m increases, although we obtain better quality approximations, the computation time increases. Since there is an inverse relationship between quality and performance, in this work we use values of $m = 2$ and 3 and results shown are averaged between these two parameter values. We see in practice that we obtain high quality results from this setting.

5 Results

We test our algorithm on both synthetic and real-world graphs. For synthetic networks, we test two types: preferential attachment and small-world. The preferential attachment graphs are built using the Barabási-Albert model [1] and possess a scale-free degree distribution. The graph is created by adding vertices one by one. The model takes two parameters: n and d, where n is the number of vertices in the graph and d is the number of edges each new vertex is given when it is first inserted into the graph. To create a scale-free distribution, edges of the newly inserted vertex connect to vertices already in the network with a probability proportional to the degree of the existing vertices. Small-world networks are build using the Watts-Strogatz model [30]. This model produces graphs with high levels of clustering as seen in real networks and with small graph diameter (the small-world property). This model takes three parameters: n, d, and p, where n is the number of vertices in the graph, which are arranged in a ring and connected to their d nearest neighbors. Each vertex is then independently considered and with probability p an edge is placed between the vertex and a randomly chosen vertex. Here, we fix p at 0.1. For both types of graphs (Barabási-Albert and Watts-Strogatz) we use values of $n = 1000, 2000$, and 3000 and vary d from $1 - 10$. For real graphs, we draw from the KONECT [17] collection of datasets, listed in Table 1. All the real graphs are temporal networks, meaning the edges have timestamps associated with them.

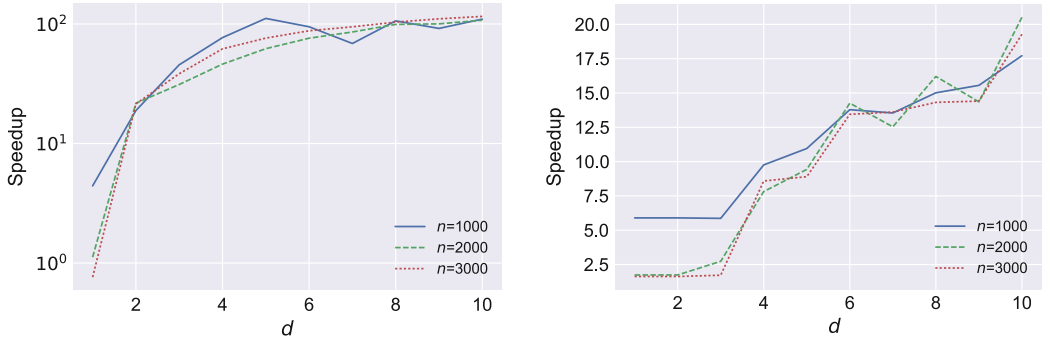
For our experiments, we insert edges in timestamped order for the real graphs and permute edges randomly for synthetic networks. To simulate a dynamic graph, we insert edges in batch sizes of 2^i for $i = 0, 1, 2, 3, 4, 5, 7$, and 9. Specifically, at each timepoint t, 2^i more edges are added to the graph. We compare the performance and quality of our dynamic algorithm to the standard static algorithm of recomputing the matrix exponential from scratch every time the underlying graph is changed. The code was implemented in Python and we use SciPy's built in expm function to calculate the matrix exponential. All experiments were performed on a 4 core Intel Xeon CPU at 2.40 GHz.

Table 1. Real graphs used in experiments.

Graph	$\|V\|$	$\|E\|$
Facebook	2,888	2,981
Power-grid	4,941	6,594
ca-HepTh	9,877	25,998
wb-cs-stanford	9,435	36,854

5.1 Synthetic Graphs

For synthetic graphs, we show results for a batch size of 1. First we measure performance of our dynamic algorithm. Let T_S denote the time taken by the static recomputation averaged over all points in time and let T_D denote the time taken by our dynamic algorithm. To measure performance, we calculate speedup as $speedup = T_S/T_D$. Values greater than 1 indicate that our dynamic algorithm is faster than a pure static recomputation. Figures 1a and b plot speedup versus d for the preferential attachment graphs and small-world graphs, respectively. The speedups for the preferential attachment graphs are several orders of magnitude higher than the corresponding small-world networks. For both types of graphs however, as the graph becomes denser (larger values of d), the speedups increase. The speedups seen can be attributed to two factors: the sparsity of ΔA and the rate of convergence of $e^{\Delta A}$ versus that of e^{A_t}. Since ΔA only consists of the edge updates at a particular time point, this matrix contains far fewer entries than that of A_t and therefore the calculations needed for the matrix exponential for ΔA will converge far quicker than those needed for the full matrix A_t as is required by the static algorithm.



(a) Preferential attachment graphs speedup.

(b) Small world graphs speedup.

Fig. 1. Speedup for synthetic graphs for batch size $2^0 = 1$.

Next we evaluate the quality of our dynamic algorithm with respect to static recomputation. Many applications in data analysis are concerned with only the highly ranked vertices in graphs [3]. Therefore to measure quality, we calculate

recall of the top k vertices for $k = 25, 50$, and 100. Let $R_S(k)$ be the set of the top k highly ranked vertices from static recomputation and $R_D(k)$ be the set of the top k vertices from our dynamic algorithm. Then recall of the top k vertices is calculated as $recall_k = |R_S(k) \cap R_D(k)|/k$. Values close to 1 indicate that our algorithm identifies a high percentage of the top ranked vertices compared to the solution from static recomputation. Tables 2 and 3 show values of the recall for the top 25, 50, and 100 highly ranked vertices for different values of d for the preferential attachment and small-world graphs, respectively. For both types of graphs we average over $n = 1000, 2000$ and 3000. We observe that the recall values for the preferential attachment graphs are higher than their small world counterparts. This can be attributed to the different degree distributions of the two types of graphs. Due to the manner of creation of the small-world networks, the topology of the network is relatively homogeneous and all vertices have essentially the same degree. In contrast, the preferential attachment graphs have hubs and a scale-free degree distribution. The difference in rankings of vertices is more likely much more prominent in graphs with a scale-free degree distribution (the preferential attachment graphs) compared to graphs with a much more homogenous degree distribution (the small-world graphs). In graphs where all vertices have a similar degree it is likely that the centrality scores themselves are also fairly similar. Since our dynamic algorithm is an approximation to the statically recomputed scores, with similar centrality scores, the rankings can themselves be easily interchanged for similarly valued vertices. Therefore it is not surprising that the recall values for the small-world graphs are lower than their preferential attachment counterparts. Furthermore, while the recall values for the preferential attachment graphs decrease as values of d increase, there is no such trend for the small world graphs, which tend to have fairly constant values of recall for different values of d.

Table 2. Recall values for preferential attachment graphs.

d	$recall_{25}$	$recall_{50}$	$recall_{100}$
1	0.88	0.88	0.88
2	0.90	0.91	0.91
3	0.88	0.88	0.89
4	0.87	0.87	0.88
5	0.85	0.85	0.86
6	0.84	0.84	0.85
7	0.82	0.83	0.83
8	0.80	0.80	0.81
9	0.77	0.77	0.79
10	0.75	0.76	0.76

Table 3. Recall values for small world graphs.

d	$recall_{25}$	$recall_{50}$	$recall_{100}$
1	0.77	0.78	0.80
2	0.77	0.78	0.80
3	0.77	0.78	0.82
4	0.78	0.80	0.83
5	0.80	0.82	0.83
6	0.77	0.80	0.84
7	0.72	0.73	0.80
8	0.73	0.78	0.80
9	0.71	0.76	0.80
10	0.68	0.72	0.78

Note again that these results are averaged over values of $m = 2$ and 3. As mentioned earlier, there is an inverse relationship between computational cost and quality of our algorithm with respect to choosing the paramater m. Specifically, as we increase the value of m, we would obtain recall values approaching closer to 1, but at a higher computational cost.

5.2 Real Graphs

Next we evaluate our dynamic algorithm on the real graphs from Table 1. In terms of performance, Fig. 2 plots the speedup versus batch size (note that both axes are on a log scale base 2 for clarity). We are able to obtain up to a 32× speedup for batch sizes larger than $2^3 = 8$ with a median speedup of about 16×, and we always have greater than a 1× speedup. As the batch size increases, the speedup obtained increases to a certain point, after which it plateaus at an average of around 16× speedup.

Next we examine the quality of our algorithm on real-world graphs. Table 4 gives the average recall values over all points in time for all batch sizes for all graphs for the top R highly ranked vertices for $R = 25, 50$, and 100, and gives the average over all batch sizes. In most cases, the average recall is over 0.75 indicating our algorithm is able to retrieve a large percentage of the highly ranked vertices compared to static recomputation. There is also a slight trend of increasing values of recall with larger batch sizes, though the average recalls over all batch sizes are fairly high. Figure 3 plots the recall over time for all the graphs for a batch size of 128, though trends for other batch sizes are similar. The x-axis simulates time as we insert more edges into the graph and the y-axis plots the recall at that point in time. The most important trend we note is that while there are occasionally dips in the recall values over time, there is no overall trend of the quality worsening over time. This indicates that at no point in time is there evidence that we need to restart our dynamic algorithm. In fact, for some

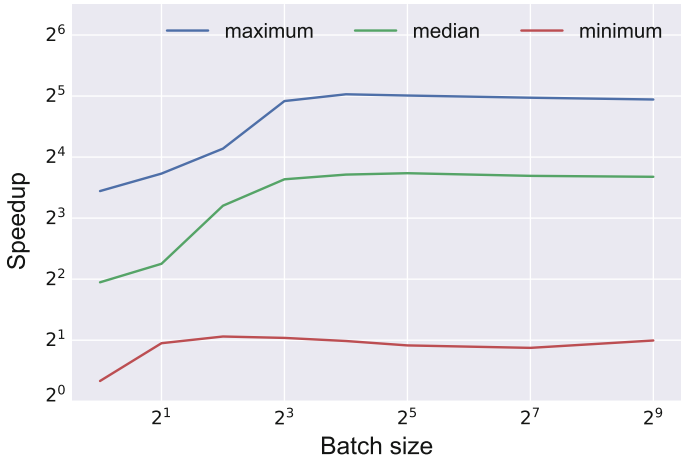


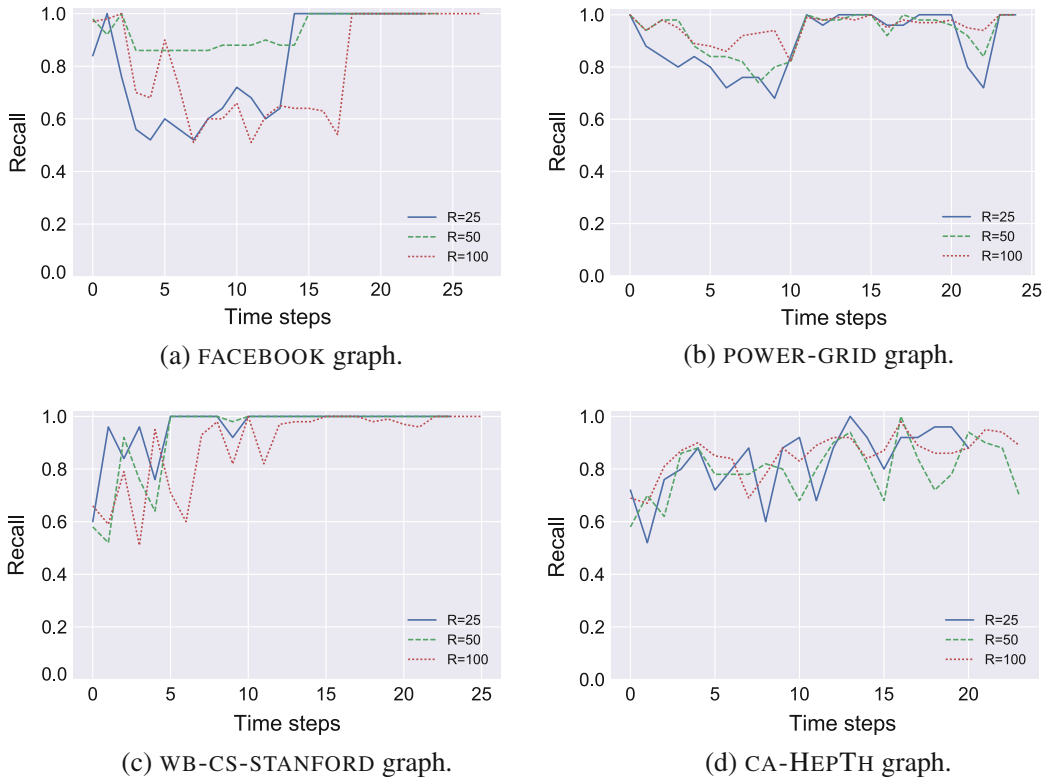
Fig. 2. Speedup versus batch size for real graphs.

Table 4. Recall for real-world graphs.

Graph	Top R	Batch size								Average
		2^0	2^1	2^2	2^3	2^4	2^5	2^7	2^9	
facebook	R = 25	0.61	0.59	0.79	0.79	0.72	0.80	0.74	0.73	0.72
	R = 50	0.83	0.81	0.93	0.93	0.93	0.93	0.89	0.98	0.90
	R = 100	0.61	0.61	0.80	0.78	0.77	0.80	0.76	0.76	0.74
power-grid	R = 25	0.83	0.86	0.86	0.86	0.87	0.89	0.89	0.92	0.87
	R = 50	0.84	0.87	0.87	0.86	0.86	0.88	0.93	0.87	0.87
	R = 100	0.86	0.88	0.87	0.88	0.88	0.90	0.95	0.92	0.89
wb-cs-stanford	R = 25	0.52	0.66	0.80	0.90	0.95	0.96	0.96	0.97	0.84
	R = 50	0.58	0.56	0.70	0.89	0.94	0.94	0.93	0.94	0.81
	R = 100	0.69	0.66	0.69	0.84	0.90	0.90	0.89	0.92	0.81
ca-HepTh	R = 25	0.68	0.63	0.79	0.86	0.88	0.89	0.83	0.76	0.79
	R = 50	0.57	0.63	0.73	0.82	0.83	0.81	0.80	0.73	0.74
	R = 100	0.60	0.72	0.77	0.84	0.86	0.86	0.85	0.79	0.79

of the graphs (WB-CS-STANFORD and CA-HEPTH) the recall actually increases over time.

Finally, in addition to recall, we examine the Kendall rank correlation coefficient (τ), a measure of the correspondence between two rankings [15]. For two $n \times 1$ vectors $\mathbf{x}$ and $\mathbf{y}$, we define P to be the number of concordant pairs (the number of elements where the ranks given by both $\mathbf{x}$ and $\mathbf{y}$ agree) and Q to be the number of discordant pairs. For example, a pair of elements (i, j) is *concordant* if both $x_i > x_j$ and $y_i > y_j$ or if both $x_i < x_j$ and $y_i < y_j$. They are *discordant* if $x_i > x_j$ and $y_i < y_j$ or if $x_i < x_j$ and $y_i > y_j$. The Kendall τ coefficient is then calculated as $\tau = \frac{P-Q}{n(n_1)/2}$. We compare the rankings given by the entire statically recomputed vector versus the vector obtained from our dynamic



(a) FACEBOOK graph. (b) POWER-GRID graph.

(c) WB-CS-STANFORD graph. (d) CA-HEPTH graph.

Fig. 3. Recall over time for different graphs for batch size $2^7 = 128$.

algorithm. Values close to 1 indicate strong agreement whereas values close to -1 indicate strong disagreement. Specifically, if the two rankings agree perfectly (they provide the same rankings for all pairs of vertices) we expect a value of 1. Similarly, if the two rankings disagree perfectly, τ would be 1 A value of 0 indicates the two rankings have no relationship to each other. Table 5 gives the values of τ for the real graphs averaged over all batch sizes and over all points in time. We note that for all real graphs tested, the value of τ is above 0 indicating that there is always agreement between the statically computed vector and dynamically computed vector. For all but one of the real graphs, the value of τ is above 0.7, indicating a strong agreement in the rankings of all the vertices.

Table 5. Values of τ for real graphs.

Graph	τ
Facebook	0.747
Power-grid	0.812
ca-HepTh	0.528
wb-cs-stanford	0.761

6 Conclusions

In this paper, we presented a new algorithm for computing the values of exponential-based centrality in dynamic graphs by studying the matrix exponential. We tested our method on both synthetic and real-world graphs and observe that our dynamic algorithm outperforms static recomputation. Additionally, the quality of our method is robust and does not decay over time, meaning that since there is no significant drift, there is no evidence that we would need to recompute the values at any point in time. Since this work compares the quality of our streaming algorithm to the exact computation of the matrix exponential (which is a computationally heavy task), the graphs used were fairly small. However, future work will consist of scaling our algorithm to larger graphs, which would include investigation of alternative methods of approximating the matrix exponential. Additionally, future work can compare rankings obtained from using the diagonal entries of the matrix exponential to row sums and observing how these rankings change over time in dynamic graphs.

Acknowledgments. Eisha Nathan is in part supported by the National Physical Science Consortium Graduate Fellowship. The work depicted in this paper was sponsored in part by the National Science Foundation under award #1339745. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

References

1. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Rev. Mod. Phys. **74**(1), 47 (2002)
2. Alsayed, A., Higham, D.J.: Betweenness in time dependent networks. Chaos, Solitons & Fractals **72**, 35–48 (2015)
3. Bauer, F., Lizier, J.T.: Identifying influential spreaders and efficiently estimating infection numbers in epidemic models: a walk counting approach. EPL Europhysics Lett. **99**(6), 68007 (2012)
4. Benzi, M., Estrada, E., Klymko, C.: Ranking hubs and authorities using matrix functions. Linear Algebra Appl. **438**(5), 2447–2474 (2013)
5. Benzi, M., Klymko, C.: Total communicability as a centrality measure. J. Complex Netw. **1**(2), 124–149 (2013)
6. Bonacich, P.: Power and centrality: a family of measures. Am. J. Sociol. **92**(5), 1170–1182 (1987)
7. Bonacich, P.: Some unique properties of eigenvector centrality. Soc. Netw. **29**(4), 555–564 (2007)
8. Braha, D., Bar-Yam, Y.: From centrality to temporary fame: dynamic centrality in complex networks. Complexity **12**(2), 59–63 (2006)
9. Estrada, E.: The structure of complex networks: theory and applications. Oxford University Press (2012)
10. Estrada, E., Higham, D.J.: Network properties revealed through matrix functions. SIAM Rev. **52**(4), 696–714 (2010)
11. Estrada, E., Rodriguez-Velazquez, J.A.: Subgraph centrality in complex networks. Phys. Rev. E **71**(5), 056103 (2005)

12. Gleich, D.F.: Pagerank beyond the web. SIAM Rev. **57**(3), 321–363 (2015)
13. Grindrod, P., Higham, D.J.: A matrix iteration for dynamic network summaries. SIAM Rev. **55**(1), 118–128 (2013)
14. Higham, N.J.: Functions of matrices: theory and computation. SIAM (2008)
15. Kendall, M.G.: A new measure of rank correlation. Biometrika **30**(1/2), 81–93 (1938)
16. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM (JACM) **46**(5), 604–632 (1999)
17. Kunegis, J.: Konect: the koblenz network collection. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 1343–1350. ACM (2013)
18. Langville, A.N., Meyer, C.D.: Updating pagerank with iterative aggregation. In: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters, pp. 392–393. ACM (2004)
19. Langville, A.N., Meyer, C.D.: A survey of eigenvector methods for web information retrieval. SIAM Rev. **47**(1), 135–161 (2005)
20. Langville, A.N., Meyer, C.D.: Who's# 1?: The Science of Rating and Ranking. Princeton University Press (2012)
21. Meyer, C.D.: Stochastic complementation, uncoupling markov chains, and the theory of nearly reducible systems. SIAM Rev. **31**(2), 240–272 (1989)
22. Moler, C., Van Loan, C.: Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. SIAM Rev. **45**(1), 3–49 (2003)
23. Newman, M.: Networks: An Introduction. Oxford University Press (2010)
24. Newman, M.E.: The structure and function of complex networks. SIAM Rev. **45**(2), 167–256 (2003)
25. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web (1999)
26. Pan, R.K., Saramäki, J.: Path lengths, correlations, and centrality in temporal networks. Phys. Rev. E **84**(1), 016105 (2011)
27. Stewart, W.J.: Introduction to the Numerical Solutions of Markov Chains. Princeton University Press, Princeto (1994)
28. Taylor, D., Myers, S.A., Clauset, A., Porter, M.A., Mucha, P.J.: Eigenvector-based centrality measures for temporal networks. Multiscale Modeling Simul. **15**(1), 537–574 (2017)
29. Trotter, H.F.: On the product of semi-groups of operators. Proc. Am. Math. Soc. **10**(4), 545–551 (1959)
30. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-worldnetworks. Nature **393**(6684), 440–442 (1998)