A Dynamic Algorithm for Updating Katz Centrality in Graphs

Eisha Nathan and David A. Bader
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
enathan3@gatech.edu, bader@cc.gatech.edu

***Abstract*—Many large datasets from a variety of fields of research can be represented as graphs. A common query is to identify the most important, or highly ranked, vertices in a graph. Centrality metrics are used to obtain numerical scores for each vertex in the graph. The scores can then be translated to rankings identifying relative importance of vertices. In this work we focus on Katz Centrality, a linear algebra based metric. In many real applications, since data is constantly being produced and changed, it is necessary to have a dynamic algorithm to update centrality scores with minimal computation when the graph changes. We present an algorithm for updating Katz Centrality scores in a dynamic graph that incrementally updates the centrality scores as the underlying graph changes. Our proposed method exploits properties of iterative solvers to obtain updated Katz scores in dynamic graphs. Our dynamic algorithm improves performance and achieves speedups of over two orders of magnitude compared to a standard static algorithm while maintaining high quality of results.**

I. Introduction

Graphs are a natural representation for modeling relationships between entities, in web traffic, financial transactions, computer networks, or society [1]. In real-world networks today, new data is constantly being produced, leading to the notion of dynamic graphs. Dynamic graph data can represent the changing relationships in networks. For example, consider a graph modeling relationships on Facebook, where vertices are people and edges exist between two vertices if the corresponding people are friends on Facebook. As new friendships are formed and old ones deleted, the corresponding graph will change over time to reflect these new relationships. The field of dynamic graph analysis is a rapidly growing field. Specifically, the identification of central vertices in an evolving network is a fundamental problem in network analysis [2]. Development of dynamic algorithms for updating centrality measures in graphs is therefore an important research problem. In this work, we present a new algorithm for updating Katz Centrality scores in dynamic graphs.

ASONAM '17, July 31 - August 03, 2017, Sydney, Australia

ACM ISBN 978-1-4503-4993-2/17/07?/$15.00
http://dx.doi.org/10.1145/3110025.3110034

Katz Centrality is a metric that measures the affinity between vertices as a weighted sum of the walks between them and penalizes long walks in the network by a user-chosen factor α [3]. The linear algebraic formulation of Katz Centrality lends itself to a dynamic algorithm based in a numerical linear algebra environment using iterative solvers. We develop an algorithm that updates Katz scores as new connections are made in the network.

A. Contributions

Previous algorithms for dynamic centrality measures have focused on algorithmically updating the centrality metric by using structural properties of the graph or by maintaining additional data structures. However, there has been little work in updating a centrality metric from a linear algebra standpoint. We present a new method of incrementally computing Katz Centrality scores in a dynamic graph that is faster than recomputing centrality scores from scratch every time the graph is updated. Furthermore, our algorithm returns high quality results that are similar to results obtained with a simple static recomputation method. To our knowledge, this is the first work on dynamic Katz Centrality.

B. Related Work

Betweenness and closeness centrality are two popular graph metrics in network analysis for identifying the most important vertices in a graph using shortest-path calculations, with specific applications in network stability, traffic predictions, and social network analysis [4], [5]. A dynamic algorithm to update both betweenness and closeness calculations simultaneously after receiving edge updates to the graph is proposed in [6]. They use the calculations performed in previous timesteps to avoid performing unnecessary calculations in the current timestep. In [7], an incremental algorithm for closeness centrality is developed that exploits specific network topological properties, such as shortest-distance distributions and the existence of vertices with identical neighborhoods. Finally, [8] proposes an incremental algorithm for updating betweenness centrality values by maintaining additional data structures to store previously computed values.

There has also been previous work in incrementally updating linear algebra based centrality measures. In [9], the eigenvalue formulation of PageRank is used to update the

ranking vector using the power method. By using exact [10] and approximate [11] aggregation techniques, the transition matrix is efficiently updated after the graph is changed. However, this approach requires accessing the entire graph, which can be very memory-inefficient. Incremental computation of PageRank, personalized PageRank, and other random walk based methods on large-scale dynamic networks is examined in [12] using Monte Carlo methods by maintaining a small number of short random walk segments starting at each node in the graph. For the case of identifying the top k vertices, these methods are able to provide highly accurate estimates of the centrality values for the top vertices, but smaller values in the personalized case are nearly identical and therefore impossible to tell apart. In [13], an algorithm for updating PageRank values in dynamic graphs by only using sparse updates to the residual is presented. To the authors' knowledge, there is no prior work in developing a dynamic algorithm for updating Katz scores in dynamic graphs.

We present our algorithm for updating Katz Centrality in dynamic graphs in Section II. Section III provides an analysis of our method on both synthetic and real-world networks with respect to performance and quality. In Section IV we conclude.

II. Methodology

Many data analysis problems are phrased as numerical problems for a more tractable solution [14]. In this work we use a linear algebra based method to compute Katz Centrality to obtain updated centrality scores on the vertices of a dynamic graph.

A. Definitions

Let $G = (V, E)$ be a graph, where V is the set of n vertices and E the set of m edges. Denote the $n \times n$ adjacency matrix A of G as

$$A(i,j) = \begin{cases} 1, & \text{if } (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

We use undirected, unweighted graphs so $\forall i, j, A(i,j) = A(j,i)$ and all edge weights are 1. A dynamic graph can change over time due to edge insertions and deletions and vertex additions and deletions. As a graph changes, we can take snapshots of its current state. We denote the current snapshot of the dynamic graph G and corresponding adjacency matrix A at time t by $G_t = (V_t, E_t)$ and A_t respectively. In this work, the vertex set is constant over time so $\forall t, V_t = V$, and we deal only with edge insertions, although our algorithm is easily generalized to edge deletions. Given edge updates to the graph, we write the new adjacency matrix at time $t+1$ as $A_{t+1} = A_t + \Delta A$, where ΔA represents the new edges being added into the graph.

Katz Centrality scores count the number of weighted walks in a graph starting and ending at each vertex and penalize longer walks with a user-chosen parameter α, where $\alpha \in (0, 1/\|A\|_2)$. Formally, the Katz centrality of vertex i is given by $\mathbf{e}_i^T \sum_{k=1}^{\infty} \alpha^{k-1} A^k \mathbf{1}$, where α is the user-chosen parameter, $\mathbf{e}_i$ is the ith canonical basis vector, and $\mathbf{1}$ is the $n \times 1$ vector of all ones. In practice the Neumann formula [15] is employed to turn this series into a linear solve and we compute the Katz Centrality of all vertices in the graph as the $n \times 1$ vector $\mathbf{c} = \sum_{k=1}^{\infty} \alpha^{k-1} A^k \mathbf{1} = A(I - \alpha A)^{-1}\mathbf{1}$. We set $\alpha = 0.85/\|A\|_2$ as in [16].

B. Iterative Methods

Directly solving for the exact Katz Centrality scores $\mathbf{c}$ is on the order of $\mathcal{O}(n^3)$ and quickly becomes very expensive as n grows large. In practice, we use iterative methods to obtain an approximation which costs $\mathcal{O}(m)$ provided the number of iterations is not very large. Many real-world graphs are sparse and $m \ll n^2$ [17]. Iterative methods approximate the solution $\mathbf{x}$ to a linear system $M\mathbf{x} = \mathbf{b}$, given M and $\mathbf{b}$ by starting with an initial guess $\mathbf{x}^{(0)}$ and improving the current guess with each iteration until some stopping criterion is reached. This stopping criterion can be a predetermined number of iterations, a desired level of accuracy, or some application-specific terminating criterion. At each iteration k of the iterative solver we obtain a new approximation $\mathbf{x}^{(k)}$. It is fairly common to assume a starting vector $\mathbf{x}^{(0)}$ as the all zeros vector, although in practice, any starting vector can be chosen. In this work, we terminate the solver when the residual reaches a preset tolerance of 10^{-4}, which is sufficient to obtain convergence of scores [13]. The residual at the kth iteration is defined as $\mathbf{r}^{(k)} = \mathbf{b} - M\mathbf{x}^{(k)}$ and is a measure of how close the current solution $\mathbf{x}^{(k)}$ is to solving the system $M\mathbf{x} = \mathbf{b}$. We let $M = I - \alpha A$, so we solve the linear system $M\mathbf{x} = \mathbf{1}$ for $\mathbf{x}$ using an iterative method and then obtain the Katz scores using a matrix-vector multiplication in $\mathcal{O}(m)$ as $\mathbf{c} = A\mathbf{x}$. The residual at iteration k is defined as $\mathbf{r}^{(k)} = \mathbf{1} - M\mathbf{x}^{(k)}$. The iterative method we use here is the Jacobi algorithm [18] outlined in Algorithm 1. Here, D is the matrix consisting of the diagonal entries from M and R is the matrix of all off-diagonal entries of M.

Algorithm 1 Solve $M\mathbf{x} = \mathbf{b}$ to tolerance tol using Jacobi algorithm.

```
1:  procedure JACOBI(M, b, tol)
2:      k = 0
3:      x^(0) = 0
4:      r^(0) = b − M x^(0)
5:      D = diag(M)
6:      R = M − D
7:      while ||r^(k)||_2 > tol do
8:          x^(k+1) = D^−1 (R x^(k) + b)        ▷ Update vector
9:          r^(k+1) = b − M x^(k+1)             ▷ Next residual
10:         k+ = 1
        return x^(k+1)
```

Our dynamic algorithm is also motivated by principles of iterative refinement, another iterative method that adds a correction to the current guess to obtain a more accurate approximation [19]. To compute the solution $\mathbf{x}$ to the linear system $M\mathbf{x} = \mathbf{b}$, iterative refinement repeatedly performs the following steps at each iteration k.

1) Compute residual $\mathbf{r}^{(k)} = \mathbf{b} - M\mathbf{x}^{(k)}$
2) Solve system $M\mathbf{d}^{(k)} = \mathbf{r}^{(k)}$ for correction $\mathbf{d}^{(k)}$
3) Add correction to obtain new solution $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)}$

Note that we can use any other iterative method to solve the system in Step 2 above.

C. Static Algorithm

Given edge updates to the graph, the static algorithm to recompute the Katz Centrality scores in the updated graph first calculates $\mathbf{x}$ from scratch using an iterative method and then $\mathbf{c}$ using a single matrix-vector multiplication. This procedure is given in Algorithm 2 to obtain the new solution $\mathbf{c}_{t+1}$ at time $t+1$ given edge updates ΔA to the graph. After a batch of edges has been inserted into the network, the adjacency matrix is updated to A_{t+1} and the vector $\mathbf{x}_{t+1}$ is recomputed using the Jacobi method from Algorithm 1.

Algorithm 2 Solve for $\mathbf{c}_{t+1}$ at time $t+1$ given new edge updates ΔA.

1: **procedure** STATIC_KATZ($A_t, \Delta A$)
2: $A_{t+1} = A_t + \Delta A$ ▷ Updated adjacency matrix
3: $M_{t+1} = I - \alpha A_{t+1}$ ▷ New linear system
4: $\mathbf{x}_{t+1}$ = JACOBI($M_{t+1}, \mathbf{1}, 10^{-4}$) ▷ Recomputed vector
5: $\mathbf{c}_{t+1}$ = $A_{t+1}\mathbf{x}_{t+1}$ ▷ New Katz scores
6: **return** $\mathbf{c}_{t+1}$

Since calculating $\mathbf{c}_t$ given $\mathbf{x}_t$ at any timepoint t is one matrix-vector multiplication and can be done in $\mathcal{O}(m)$, this is not the bottleneck of the static algorithm. As more data is added to the graph, the number of iterations taken to update $\mathbf{x}_{t+1}$ increases. Therefore, pure recomputation becomes increasingly expensive as the graph size increases. We thus focus the development of our dynamic algorithm on limiting the number of iterations taken to obtain the updated vector $\mathbf{x}_{t+1}$. Calculating $\mathbf{c}$ is the same in the static and dynamic algorithm and so for the rest of the paper we focus our discussions on the vector $\mathbf{x}$.

D. Dynamic Algorithm

In many low-latency applications, the number of edge updates, or equivalently, the size of ΔA, is significantly smaller than the size of the entire graph A. If the change ΔA is small relative to the size of the graph, the new graph will be similar to the old graph. It follows that the new solution $\mathbf{x}_{t+1}$ at time $t+1$ might be similar to the old solution $\mathbf{x}_t$ at time t. This is the intuition behind our dynamic algorithm. Figure 1 plots the differences between subsequent solutions each time the graph changes. The x-axis simulates time as more edges are being added into the graph. We insert 1000 edges into the FACEBOOK graph at each time step. The y-axis is the 2-norm difference between solutions at consecutive timepoints, $\|\mathbf{x}_{t+1} - \mathbf{x}_t\|_2$. Since the Katz scores themselves can be as high as 10^4, a difference of 10^{-1} across insertions of edges over time is relatively small. This indicates that the solutions themselves are not very different, suggesting that the static algorithm of recomputing the centrality metric from scratch is doing a lot of unnecessary work. Our dynamic algorithm therefore only targets places in the vector that are affected by updates to the graph.

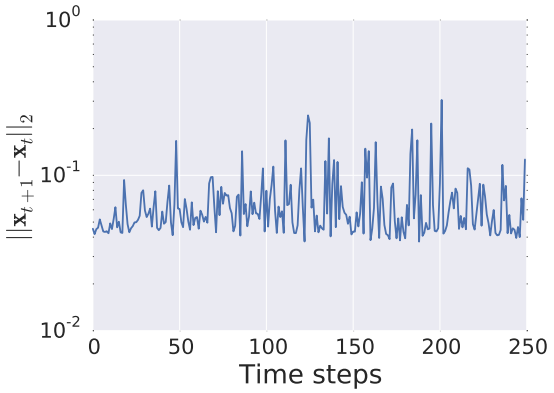


Fig. 1: Difference in consecutive solutions over time. Small changes in solutions suggest a dynamic algorithm could work by applying incremental updates to previous solutions.

Suppose we have the solution $\mathbf{x}_t$ for the adjacency matrix A_t at a specific timepoint t. We solve for the new solution at time $t+1$ as $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{\Delta x}$. Our dynamic algorithm computes the correction $\mathbf{\Delta x}$, the difference in the solutions at timepoints t and $t+1$, using principles of iterative refinement. The full dynamic algorithm is given in Algorithm 4 and calculates the updated vector $\mathbf{x}_{t+1}$ given the old solution $\mathbf{x}_t$ and the edge updates ΔA.

Algorithm 3 Solve for $\mathbf{x}_{t+1}$ at time $t+1$ given previous solution $\mathbf{x}_t$ at time t and new edge updates ΔA.

1: **procedure** DYNAMIC_KATZ($A_t, \mathbf{x}_t, \Delta A$)
2: $\mathbf{r}_t = \mathbf{1} - (I - \alpha A_t)\mathbf{x}_t = \mathbf{1} - \mathbf{x}_t + \alpha A_t$
3: $A_{t+1} = A_t + \Delta A$
4: $\widetilde{\mathbf{r}}_{t+1} = \mathbf{r}_t + \alpha \Delta A \mathbf{x}_t$
5: $\mathbf{\Delta x}$ = JACOBI($I - \alpha A_{t+1}, \widetilde{\mathbf{r}}_{t+1}, 10^{-4}$)
6: $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{\Delta x}$
7: $\mathbf{\Delta r} = \alpha \Delta A \mathbf{x}_t - (I - \alpha A_{t+1})\mathbf{\Delta x}$
8: $\mathbf{r}_{t+1} = \mathbf{r}_t + \mathbf{\Delta r}$
9: **return** $\mathbf{x}_{t+1}$

First in line 2 we calculate the current residual $\mathbf{r}_t$, which is easily obtained given the current snapshot of the graph A_t and solution $\mathbf{x}_t$ at time t. In line 3, we form the new snapshot of the graph A_{t+1} using the new batches of edges that are being inserted into the graph. Since we use the old solution as a starting point for the new solution, we first measure how close the old solution is to solving the system for the new graph. We do so by introducing the concept of an "approximate residual" denoted as $\widetilde{\mathbf{r}}_{t+1}$. This can be written in terms of the current residual at time t, $\mathbf{r}_t = \mathbf{1} - M_t\mathbf{x}_t$, edge updates ΔA, and the old solution $\mathbf{x}_t$. This is calculated in line 4. We derive $\widetilde{\mathbf{r}}_{t+1}$

in Equation 1.

$$\begin{aligned}\widetilde{\mathbf{r}}_{t+1} &= \mathbf{1} - M_{t+1}\mathbf{x}_t \\ &= \mathbf{1} - (I - \alpha A_{t+1})\mathbf{x}_t \\ &= \mathbf{1} - \mathbf{x}_t + \alpha A_{t+1}\mathbf{x}_t \\ &= \mathbf{1} - \mathbf{x}_t + \alpha A_t\mathbf{x}_t - \alpha A_t\mathbf{x}_t + \alpha A_{t+1}\mathbf{x}_t \\ &= \mathbf{r}_t + \alpha(A_{t+1} - A_t)\mathbf{x}_t \\ &= \mathbf{r}_t + \alpha\Delta A\mathbf{x}_t \end{aligned} \quad (1)$$

We then use the approximate residual $\widetilde{\mathbf{r}}_{t+1}$ to solve a linear system for the correction $\mathbf{\Delta x}$. Solved exactly, this linear system will give the same scores as static recomputation but solved to the same tolerance as used earlier (10^{-4}), it will provide a good quality approximation of the updated centrality scores. This is calculated in line 5. Since the approximation residual $\widetilde{\mathbf{r}}_{t+1}$ measures how close the current solution is to the solution of the updated system, we use $\widetilde{\mathbf{r}}_{t+1}$ to solve for the correction $\mathbf{\Delta x}$ using principles of iterative refinement:

$$\begin{aligned}(I - \alpha A_{t+1})\mathbf{\Delta x} &= \widetilde{\mathbf{r}}_{t+1} = \mathbf{r}_t + \alpha\Delta A\mathbf{x}_t \\ \mathbf{\Delta x} - \alpha A_{t+1}\mathbf{\Delta x} &= \mathbf{r}_t + \alpha\Delta A\mathbf{x}_t\end{aligned}$$

We can turn this into an iterative update:

$$\mathbf{\Delta x}^{(k+1)} = \alpha A_{t+1}\mathbf{\Delta x}^{(k)} + \alpha\Delta A\mathbf{x}_t + \mathbf{r}_t$$

This formulation lends itself quite nicely to using the Jacobi algorithm. In line 6 we calculate the new solution $\mathbf{x}_{t+1}$ using the old solution $\mathbf{x}_t$ and the calculated correction $\mathbf{\Delta x}$. After updating the solution from time t to the solution at $t+1$, lines 7 and 8 update the residual between these two timepoints. We do so by calculating $\mathbf{\Delta r}$, the difference in the two residuals at time t and $t+1$. The residual $\mathbf{r}_{t+1}$ at time $t+1$ measures the correctness of the updated solution $\mathbf{x}_{t+1}$. We write the new residual $\mathbf{r}_{t+1}$ in terms of the old residual $\mathbf{r}_t$ to obtain the difference between the two as $\mathbf{\Delta r}$.

$$\begin{aligned}\mathbf{r}_{t+1} &= \mathbf{1} - (I - \alpha A_{t+1})\mathbf{x}_{t+1} \\ &= \mathbf{1} - (I - \alpha A_{t+1})(\mathbf{x}_t + \mathbf{\Delta x}) \\ &= \mathbf{1} - (I - \alpha A_{t+1})\mathbf{x}_t - (I - \alpha A_{t+1})\mathbf{\Delta x} \\ &= \widetilde{\mathbf{r}}_{t+1} - (I - \alpha A_{t+1})\mathbf{\Delta x} \\ &= \mathbf{r}_t + \alpha\Delta A\mathbf{x}_t - (I - \alpha A_{t+1})\mathbf{\Delta x} \\ &= \mathbf{r}_t + \mathbf{\Delta r} \\ \therefore \mathbf{\Delta r} &= \alpha\Delta A\mathbf{x}_t - (I - \alpha A_{t+1})\mathbf{\Delta x}\end{aligned}$$

III. Results

We test our method of updating Katz Centrality scores in dynamic graphs on synthetic and real-world networks. For synthetic networks, we use R-MAT graphs [20]. An R-MAT generator creates scale-free networks designed to simulate real-world networks. Consider an adjacency matrix: the matrix is subdivided into four quadrants, where each quadrant has a different probability of being selected. Once a quadrant is selected, this quadrant is recursively subdivided into four subquadrants and using the previous probabilities, we select one of the subquadrants. This process is repeated until we arrive at a single cell in the adjacency matrix. An edge is assigned between the two vertices making up that cell. For real-world networks, we use five graphs taken from the KONECT collection [21]. The five datasets used are given in Table I and comprise a mixture of citation and social networks. To have a baseline for comparison, every time we update the centrality scores using our dynamic algorithm, we recompute the centrality vector statically using Algorithm 2. Denote the vector obtained by static recomputation by $\mathbf{x}_R$ and the vector obtained by our dynamic algorithm by $\mathbf{x}_S$. We create an initial graph G_0 using the first half of edges, which provides a starting point for both the dynamic and static algorithms. To simulate a stream of edges in a dynamic graph, we insert the remaining edges in batches of size b and apply both algorithms. If the edges in the graphs have timestamps associated with them, we insert them in timestamped order (denoted *Temporal* in Table I), otherwise we permute the edges randomly (denoted *Random*). We use batch sizes of $b = 1, 10, 100$, and 1000. This section provides results on our algorithm with respect to performance and quality on synthetic and real graphs.

Algorithm 4 Solve for $\mathbf{x}_{t+1}$ at time $t+1$ given previous solution $\mathbf{x}_t$ at time t and new edge updates ΔA.

1: **procedure** DYNAMIC_KATZ($A_t, \mathbf{x}_t, \Delta A$)
2: $\quad\mathbf{r}_t = \mathbf{1} - (I - \alpha A_t)\mathbf{x}_t = \mathbf{1} - \mathbf{x}_t + \alpha A_t$
3: $\quad A_{t+1} = A_t + \Delta A$
4: $\quad\widetilde{\mathbf{r}}_{t+1} = \mathbf{r}_t + \alpha\Delta A\mathbf{x}_t$
5: $\quad\mathbf{\Delta x}$ = JACOBI($I - \alpha A_{t+1}, \widetilde{\mathbf{r}}_{t+1}, 10^{-4}$)
6: $\quad\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{\Delta x}$
7: $\quad\mathbf{\Delta r} = \alpha\Delta A\mathbf{x}_t - (I - \alpha A_{t+1})\mathbf{\Delta x}$
8: $\quad\mathbf{r}_{t+1} = \mathbf{r}_t + \mathbf{\Delta r}$
9: **return** $\mathbf{x}_{t+1}$

Graph	$\lvert V\rvert$	$\lvert E\rvert$	Edge Ordering
cit-HepPth	34 546	421 578	Temporal
facebook	42 390	876 993	Temporal
slashdot	51 083	140 778	Temporal
email-EuAll	265 214	420 045	Random
twitter	465 017	834 797	Random

TABLE I: Graphs used in experiments. Columns are graph name, number of vertices, number of edges, and edge ordering.

A. Synthetic Graphs

We generate graphs with the number of vertices as a power of 2, ranging from 2^{10} to 2^{14} and vary the average degree of the graphs from 10 to 50. For each total number of vertices and average degree, five graphs are created and tested and the results shown are averaged over these five trials. All results shown for the synthetic cases use a batch size of 1, meaning after we create the initial graph G_0, we sequentially insert the remaining 1/2 of edges. The results for other batch sizes are similar.

The primary motivation behind a dynamic approach is to prune any unnecessary work in the static algorithm to

obtain a faster method of producing the ranking vector for dynamic graphs. Therefore, we evaluate the performance of the dynamic algorithm in terms of speedup compared to the static algorithm. For a particular timepoint after inserting a batch of edges, let T_R and T_S denote the time taken by static recomputation and our dynamic method respectively. Since we are using iterative methods to calculate the centrality vectors, we also evaluate the performance of the dynamic algorithm with respect to the speedup in number of iterations. Let I_R and I_S denote the number of iterations taken by static recomputation and our dynamic algorithm respectively. We calculate speedup in time as $speedup(time) = \frac{T_R}{T_S}$ and speedup in iterations as $speedup(iterations) = \frac{I_R}{I_S}$.

Average degree	*10*	*20*	*30*	*40*	*50*
n = 1024	1.75×	1.95×	2.15×	2.44×	2.7×
n = 2048	1.98×	2.39×	2.7×	3.14×	3.56×
n = 4096	2.42×	3.12×	3.62×	4.3×	5.08×
n = 8192	3.35×	4.32×	5.25×	6.41×	7.46×
n = 16384	4.63×	6.26×	7.64×	9.15×	10.46×

TABLE II: Speedup in time for R-MAT graphs.

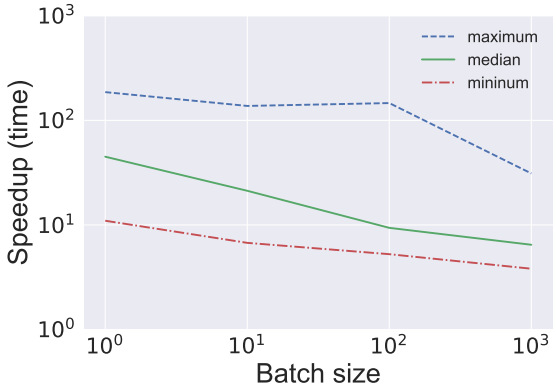
Average degree	*10*	*20*	*30*	*40*	*50*
n = 1024	4.89×	5.29×	5.6×	6.01×	6.38×
n = 2048	5.12×	5.77×	6.27×	6.73×	7.12×
n = 4096	5.34×	6.2×	6.69×	7.24×	7.66×
n = 8192	5.81×	6.52×	7.18×	7.77×	8.25×
n = 16384	6.0×	6.89×	7.62×	8.29×	8.72×

TABLE III: Speedup in iterations for R-MAT graphs.

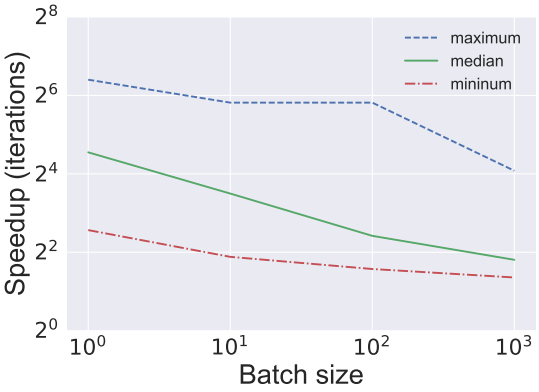
Tables II and III give the average speedup in time and iterations respectively for R-MAT graphs. As we increase the average degree in the graph, the speedups in time and iterations obtained are larger. Additionally, we see greater speedups for graphs with larger values of n. The dynamic algorithm likely has more of an effect for larger graphs because there is more work to be done for larger graphs with the static algorithm. Unlike the static algorithm, our dynamic algorithm only traverses parts of the graph where updates have occurred.

B. Real Graphs

Next we show results on real graphs (from Table I). Figures 2a and 2b plot the speedup w.r.t time and iterations versus batch size respectively, comparing our dynamic algorithm against static recomputation. We show the maximum, median, and minimum speedup over the 5 real graphs. Note that the y-axis in Figure 2a is on a log scale with base 10 and the y-axis in Figure 2b is on a log scale with base 2 for clarity. In Figure 2a we see that our dynamic algorithm can be over two orders of magnitude faster for a batch size of 1 than both static recomputation approaches. The median speedup in time is about 50× for a batch size of 1 and even for a batch size of 1000 edges we always have greater than a 1× speedup. Figure 2b shows that we can obtain over an 80× speedup in iterations for a batch size of 1. This is especially significant because the static method can take hundreds or thousands of iterations in some cases, so our algorithm would provide large savings of resources in many applications. Finally, we see a greater speedup in both time and iterations for the smaller batch sizes of 1 and 10. This is because as the batch size increases, the dynamic algorithm nears the work of a static algorithm. This shows that the dynamic approach is most useful for monitoring applications where the rankings must be updated after only a small number of data changes.



(a) Average speedup over time.



(b) Average speedup in iterations over time.

Fig. 2: Performance results for five real networks for different batch sizes.

We have shown that we are able to achieve results faster using a dynamic algorithm compared to static recomputation every time the graph changes when calculating centrality scores in dynamic networks. However, it is also important to ensure that the centrality scores returned by the dynamic algorithm are similar to those returned by the static algorithm. To measure the error between the statically computed vector $\mathbf{x}_R$ and dynamically computed vector $\mathbf{x}_S$, we compute the pointwise difference as $error = \|\mathbf{x}_R - \mathbf{x}_S\|_\infty$. Table IV gives the error averaged over all time points for each batch size and all the different graphs. We see that regardless of the batch size, the dynamic algorithm is able to maintain similar quality

Graph	Batch size 1	10	100	1000
cit-HepPh	2.08e-05	2.33e-05	1.26e-05	6.35e-06
facebook	8.61e-05	1.08e-04	1.18e-04	6.71e-05
slashdot	5.42e-05	7.56e-05	7.80e-05	7.85e-05
email-EuAll	1.36e-05	4.32e-06	3.88e-06	3.87e-06
twitter	2.57e-05	1.73e-05	1.98e-05	1.96e-05

TABLE IV: Average error for different batch sizes.

to that of static recomputation. We obtain an average error of about 10^{-4}. The error we see between $\mathbf{x}_R$ and $\mathbf{x}_S$ is negligible because the values in the ranking vector itself can be as large as 10^4. Additionally, we terminate the solver when we get to a residual norm of 10^{-4}, meaning that we are not likely to obtain much more precise solutions than this. Finally, Figure 3 plots the error for different graphs over time for a batch size of $b = 10$, though the results for different batch sizes are similar. We sample at 30 evenly spaced time points and see that the quality does not worsen over time, showing little to no increase in the error. This shows that the dynamic algorithm is robust to many edge insertions and at no point in time is there evidence that we would need to restart the dynamic algorithm.

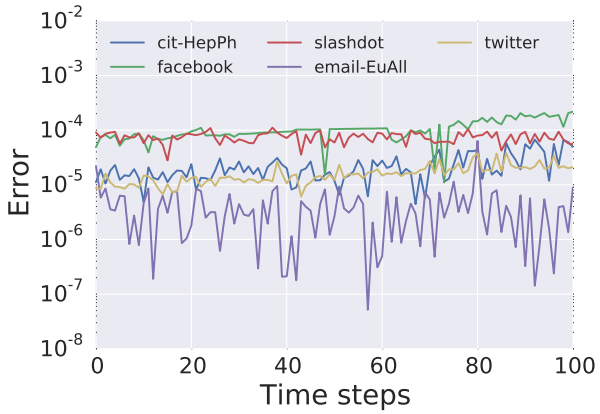


Fig. 3: Error over time for all graphs for batch size $b = 10$.

IV. Conclusion

We have presented a new algorithm that incrementally updates the Katz Centrality scores when the underlying graph changes. Our dynamic algorithm is faster than statically recomputing the centrality scores every time the graph changes, and the performance improvement is greatest when low latency updates are required. However, our approach is still faster than recomputing from scratch even for large batch insertions of edges into the graph. Our dynamic algorithm returns scores that are within negligible error of the scores returned by static recomputation. We have shown that the quality of the scores using our dynamic algorithm does not deteriorate over time. Future work will address updating the scores in a personalized setting if we desire the averaged Katz scores for all vertices from a specific seed set of vertices of interest and will extend the algorithm to deal with insertion and deletion of vertices.

Acknowledgement

This work is in part supported by a graduate fellowship from the National Physical Science Consortium.

References

[1] M. Benzi and C. Klymko, "A matrix analysis of different centrality measures," *arXiv preprint arXiv:1312.6722*, 2014.
[2] M. Benzi, E. Estrada, and C. Klymko, "Ranking hubs and authorities using matrix functions," *Linear Algebra and its Applications*, vol. 438, no. 5, pp. 2447–2474, 2013.
[3] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.
[4] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.
[5] A. Bavelas, "Communication patterns in task-oriented groups," *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, 1950.
[6] W. Wei and K. Carley, "Real time closeness and betweenness centrality calculations on streaming network data," 2014.
[7] A. E. Sariyuce, K. Kaya, E. Saule, and U. V. Catalyurek, "Incremental algorithms for closeness centrality," in *Big Data, 2013 IEEE International Conference on*. IEEE, 2013, pp. 487–492.
[8] O. Green, R. McColl, and D. A. Bader, "A fast algorithm for streaming betweenness centrality," in *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*. IEEE, 2012, pp. 11–20.
[9] A. N. Langville and C. D. Meyer, "Updating pagerank with iterative aggregation," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, 2004, pp. 392–393.
[10] C. D. Meyer, "Stochastic complementation, uncoupling markov chains, and the theory of nearly reducible systems," *SIAM review*, vol. 31, no. 2, pp. 240–272, 1989.
[11] W. J. Stewart, *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press, 1994.
[12] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank," *Proceedings of the VLDB Endowment*, vol. 4, no. 3, pp. 173–184, 2010.
[13] J. Riedy, "Updating pagerank for streaming graphs," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 2016, pp. 877–884.
[14] E. Kokiopoulou, J. Chen, and Y. Saad, "Trace optimization and eigenproblems in dimension reduction methods," *Numerical Linear Algebra with Applications*, vol. 18, no. 3, pp. 565–602, 2011.
[15] D. Werner, *Funktionalanalysis*. Springer, 2006.
[16] M. Benzi and C. Klymko, "Total communicability as a centrality measure," *Journal of Complex Networks*, vol. 1, no. 2, pp. 124–149, 2013.
[17] R. Albert, H. Jeong, and A.-L. Barabási, "Internet: Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.
[18] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.
[19] J. H. Wilkinson, *Rounding errors in algebraic processes*. Courier Corporation, 1994.
[20] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining." in *SDM*, vol. 4. SIAM, 2004, pp. 442–446.
[21] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 2013, pp. 1343–1350.