Parallel Computing 47 (2015) 38–50

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Behavioral clusters in dynamic graphs

James P. Fairbanks *, Ramakrishnan Kannan, Haesun Park, David A. Bader

School of Computational Science and Engineering, Georgia Institute of Technology, United States

ARTICLE INFO

Article history:
Available online 11 March 2015

Keywords:
Dynamic graph analysis
Streaming
Matrix factorization
Nonnegative Matrix Factorization (NMF)
Behavioral clusters
Low rank approximation

ABSTRACT

This paper contributes a method for combining sparse parallel graph algorithms with dense parallel linear algebra algorithms in order to understand dynamic graphs including the temporal behavior of vertices. Our method is the first to cluster vertices in a dynamic graph based on arbitrary temporal behaviors. In order to successfully implement this method, we develop a feature based pipeline for dynamic graphs and apply Nonnegative Matrix Factorization (NMF) to these features. We demonstrate these steps with a sample of the Twitter mentions graph as well as a CAIDA network traffic graph. We contribute and analyze a parallel NMF algorithm presenting both theoretical and empirical studies of performance. This work can be leveraged by graph/network analysts to understand the temporal behavior cluster structure and segmentation structure of dynamic graphs.

1. Introduction

There are many domains of data analysis that can be modeled with the graph abstraction. In particular we are interested in social networks and internet connection networks. These networks are collections of interactions occurring in complex patterns. Analyzing these patterns is essential to leveraging the information contained in these networks. Because the most important networks are the networks that are in heavy use right now, methods to understand temporal patterns in dynamic networks are important.

The availability of big data has driven an adoption of large scale statistical techniques, both classical and modern. These techniques are not immediately applicable to graph data and this leaves analysts separated from their familiar software tools. In order to connect graph analysis and statistical reasoning, we introduce vertex features which can be calculated efficiently and then analyzed using familiar large scale statistical software tools. This connection is bidirectional because statistical analysis of vertex features informs the computation of additional features. The observed difficulty of writing scalable parallel graph algorithms for scale-free and irregular graphs advises against writing inferential and mathematical code to analyze the graphs directly. In this paper we address this gap by first applying non-inferential graph code to generate vectorial data that is statistically well behaved, then applying a state of the art vectorial technique to this data, which provides insight into the original graph. A representation of this framework is presented in Fig. 1.

In the *massive streaming data analytics model* [11], we view the graph of network events as an unending stream of new edge updates. For each interval of time, we have the static graph, which represents the previous state of the network, and a sequence of edge updates that represent the events since the previous state was recorded. An update can take the form of inserting a new edge, a changing the weight of an existing edge, or a deleting an existing edge. Some networks do not

* Corresponding author.
E-mail addresses: james.fairbanks@gatech.edu (J.P. Fairbanks), rkannan@gatech.edu (R. Kannan), hpark@cc.gatech.edu (H. Park), bader@cc.gatech.edu (D.A. Bader).

http://dx.doi.org/10.1016/j.parco.2015.03.002
0167-8191/

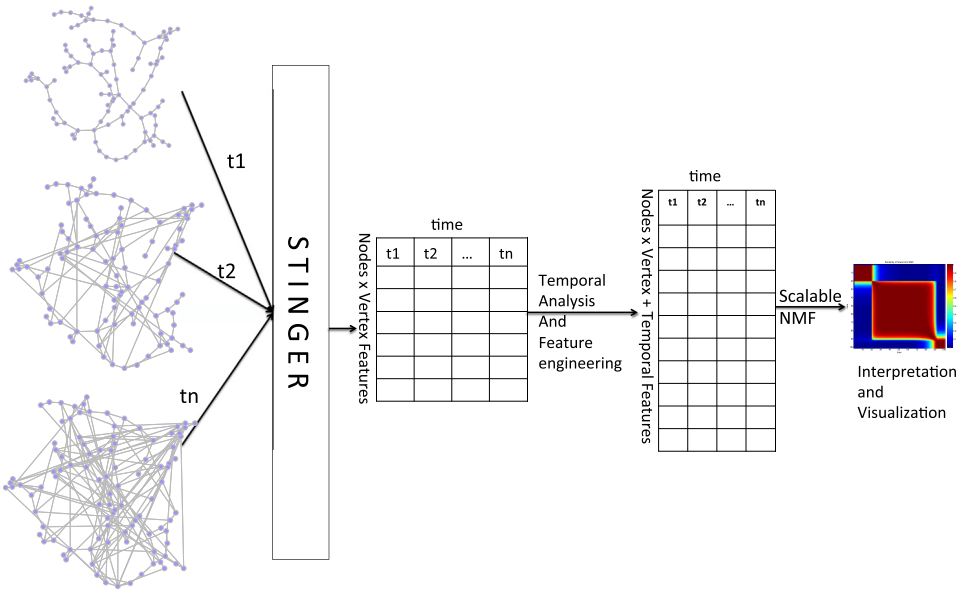


Fig. 1. Our framework combines sparse parallel graph algorithms and dense parallel linear algebra algorithms.

naturally handle deletions, for example Twitter and IP networks where messages are sent and received. In these cases we count the number of messages as the edge weight.

Early work on the theory of streaming algorithms involves summarizing data streams. In a seminal paper by Flajolet and Martin [15], the data is presented in a streaming context and the number of distinct elements must be counted. The algorithms in this field are streaming but the analysis of that data is not necessarily temporal. Feigenbaum et al. [14] have contributed to one model of streaming graph analysis by considering the "semi-streaming model" where graphs are presented "as a stream of edges in adversarial order" and the goal is to compute properties of the graph in one or sub-linearly many passes over the edge stream. This semi-streaming model takes the perspective of a fixed graph with limited access to the data. The work addresses the theoretical issues in computing solutions to "classical graph problems" with necessary approximations due to the constraints on accessing the edges.

With a dynamically changing graph where only those edges occurring in the past can be accessed, there are a new set of temporal queries to answer. Our work contributes to the analysis of modern graph problems that only appear when the edge set is fluctuating over time. We provide insight into applications of temporal data analysis techniques to large data sets that are well represented by the dynamic graph abstraction. Previous approaches to dynamic graph analysis have leveraged traditional, static graph analysis algorithms to compute an initial metric on the graph and then a final metric on the graph after all updates are processed. The underlying assumption is that the time window is large and the network changes substantially so that the entire metric must be recomputed. However, in the massive streaming data analytics model, algorithms react to much smaller changes on smaller time-scales. For example, given a graph with billions of edges, inserting 100,000 new edges might have a small impact on the overall graph, but it might have a large impact on a small subset of the graph. To accommodate this, in our approach an efficient streaming algorithm recomputes metrics on only the affected regions of the graph. For instance, when considering the betweenness centrality of vertices in IP networks each batch of edges represents approximately 36 s of internet traffic. This approach has shown large speed-ups for evaluating clustering coefficients and connected components on scale-free networks [11,13].

In this work we show that the vertex features, as explained above, can be used to generate an understanding of the vertex behavior as well as the behavior of the entire graph as a whole. The nonnegative factorization [1] of these feature matrices provides a clustering of the vertices into groups and a segmentation of the edge stream into phases, which are two important data analysis tasks. These feature matrices are broadly applicable and many applications are beyond the scope of this paper, including tensor factorizations which will provide latent feature based understanding of the three way interaction between the vertices, the different features, and the time-steps.

2. Relevant literature

Previous research has shown that Twitter posts reflect valuable information about the real world. Human events, such as breaking stories, pandemics, and crises, affect worldwide information flow on Twitter. Trending topics and sentiment

[1] Matrix factorization and low rank approximation are used interchangeably for consistency with the literature.

analysis can yield valuable insight into the global heartbeat. A number of crises and large-scale events have been extensively studied through the observation of Tweets. Hashtags, which are user-created metadata embedded in a Tweet, have been studied from the perspective of topics and sentiment. Hashtag half-life was determined to be typically less than 24 h during the London riots of 2011 [18]. The analysis of Twitter behavior following a 2010 earthquake in Chile revealed differing propagation of rumors and news stories [30]. Researchers in Japan used Twitter to detect earthquakes with high probability [33]. Twitter can be used to track the prevalence of influenza on a regional level in real time [34]. Betweenness centrality analysis applied to the H1N1 outbreak and historic Atlanta flooding in 2009 revealed highly influential tweeters in addition to commercial and government media outlets [12].

Many attempts at quantifying influence have been made. Indegree, retweets, and mentions are first-order measures, but popular users with high indegree do not necessarily generate retweets or mentions [5]. These first-order metrics are traditional database queries that do not take into account topological information. PageRank and a low effective diameter reveal that retweets diffuse quickly in the network and reach many users in a small number of hops [27]. Users tweeting URLs that were judged to elicit positive feelings were more likely to spread in the network, although predictions of which URL will lead to increased diffusion were unreliable [4].

We also know that understanding the internet as a graph can provide information about major geopolitical events. Dainotti et al. [9] conducted a thorough analysis of the internet blackout that was caused by the Egyptian government during the Arab Spring revolution. Understanding the patterns of internet connections can yield insight into the methods of agents acting to restrict the flow of information. IP network data collected by the Center for Applied Internet Data Analysis (CAIDA) with the UCSD Network Telescope was used by Dainotti et al. [8] to study the progress of a botnet composed of 3 million unique IP addresses, in scanning IP space. An analysis of IP network data and graph metrics is presented by Henderson et al. [20] focused on the relationships between different metrics. One way to handle a large graph $G = (V, E)$ is to consider it as V points in V dimensional space, where V is the vertex set, and then use dimensionality reduction techniques. These methods are used in the sketching data structures found in Ahn, Guha and McGregor [1], where linearity of the sketch gives rise to incremental and dynamic updates to the data structure. Another method to pursue dimensionality reduction is to apply a matrix approximation to the adjacency matrix directly and use the approximation to study the graph. One example of this work is Colibri-D [37].

From the statistical hypothesis testing field, [39] the authors define a hypothesis test for detecting localized increases in activity within a temporal graph. Their work emphasizes a particular generative model of the data and can be implemented with a streaming computation of the test statistic. Groups of vertices with communication density much higher than a typical group will be considered significant according to this test.

Others have used a framework similar to ours to study large graphs using vertex features including Oddball [2], which performs graph anomaly detection using local (egonet) features, and RolX [21], which uses Nonnegative Matrix Factorization (NMF) on locally extracted features. Our work is the first to use both global features and dynamic information. Here we are learning not only the roles of the vertices, but also the structure of those roles over time as the graph changes. NMF was first proposed in Paatero and Tapper [31] as Positive Matrix Factorization. But it became widely known in Lee and Seung [28] when Kullback–Leibler (KL) Divergence was used to define the closeness of the input matrix to the product of low rank factors. They produce an algorithm called Multiplicative Updates, which was widely adopted because of easy implementation. However, Gonzales and Zhang [19] proves that the Multiplicative Updates algorithm does not converge. This prompted research discovering NMF algorithms that converge to a stationery point. Towards this end, Lin [29] proposes a projected gradient descent based algorithm. Similarly, Cichocki et al. [7,6] proposes the Hierarchical Alternating Least Squares (HALS) algorithm which was independently discovered in Ho et al. [22] as rank-one residue iteration (RRI). Kim, He and Park [24] gives a unified framework based on Block Coordinate Descent that explains convergent NMF algorithms. Kim and Park [25] propose a fast greedy active set based method to solve the NMF problem and prove the convergence as well. So far, the greedy active set method is the fastest convergent algorithm in the literature. Given the sufficient literature for the NMF algorithms, we address interesting graph applications of NMF. Recently there has been a surge of interest in using NMF for social network analysis. Much of the literature on social networks using NMF assumes the input to be an adjacency matrix where each element $A_{i,j}$ represents the strength of connection between node i and node j. The most important applications of NMF on adjacency matrices are graph clustering and community detection. Kuang, Yun, and Park [26] applies NMF to the symmetric adjacency matrix of an undirected graph in order to cluster the vertices. Wang et al. [38] uses NMF to detect communities in social networks. Psorakis, Robert, Ebden, and Sheldon [32] proposes a Bayesian framework and NMF for detecting overlapping communities in networks. Yang and Leskovec [40] detects overlapping communities by applying NMF to very large social networks. Even though much of the literature on NMF for social networks pertains to clustering and community detection, Tong and Lin [36] examines a related problem of anomaly detection in social networks using a nonnegative matrix factorization with a nonnegativity constrained additive residual representing the anomalous edges. The additive component is sparse and nonnegative thus containing edges which deviate from the low rank structure of the adjacency matrix. By applying NMF to recursively extracted local vertex features, Henderson et al. [21], discover the roles of vertices within a static graph's structure. They distinguished network role discovery from network community discovery and NMF provides natural explanation of network role discovery. Computing a factorization of the adjacency matrix of a large and irregular social network will suffer the usual problems of irregular memory access, difficult load balancing, and difficult partitioning on distributed memory systems, that are familiar to those working on parallel graph algorithms for scale free

graphs. This is why our approach combines high performance graph processing algorithms with parallel dense linear algebra algorithms in order to extract insight from the graph.

2.1. Contributions

With the relevant literature in mind, we contribute a broad framework for connecting high performance graph algorithms to large scale data analysis techniques. Our method is the first to find behavioral vertex clusters in a dynamic graph. We present a feature based pipeline for dynamic graphs and apply Nonnegative Matrix Factorization (NMF) to these features, which reveals vertex clusters and phases of network activity over time.

The algorithms used in this paper present good performance in both theory and practice. Three important tasks about temporal graphs are clustering the vertices, segmenting the edge stream and visualizing changes to the graph. The low rank approximation method used in this paper provides answers to all three of these important questions.

Section 4 takes a stream of Twitter posts ("Tweets") from the time surrounding the landfall of Hurricane Sandy, a tropical storm that hit the Northern Atlantic coast of the United States, and forms a temporal social network of mentions. We compute graph metrics, including betweenness centrality and pagerank, in a streaming manner for each batch of new edges arising in the network. Statistical analysis of these features leads to the construction of additional features. In Section 5, we describe some insights into cluster structure in the CAIDA Network derived by NMF. The performance of our parallel NMF algorithm is empirically demonstrated in Section 6 and validated our theoretical analysis of parallel NMF in Section 3.3.1.

3. Foundations

In this section, we present the necessary foundations for describing our feature based graph analysis pipeline. We discuss our representation of a graph in terms of vertex features, our parallel platform for computing such features, the relevant algorithm for NMF used to detect temporal clusters, and methods for comparing the similarity of graphs over time.

3.1. Vertex Features

We define a *graph kernel* as an algorithm that builds a data structure or index on a graph.[2] We define a *vertex feature* as a function from the vertex set to the real numbers. In the context of a dynamic graph, we use temporal vertex feature to refer to any vertex feature that captures temporal or dynamic information. For example, a connected components algorithm is a graph kernel because it creates a mapping from the vertex set to the component labels. The function that assigns each vertex the size of its connected component is a vertex feature. The function that assigns to each vertex the number of new vertices in its component at time t is a temporal vertex feature. Graph kernels can be used as subroutines for the efficient computation of vertex features. Any efficient parallel implementation of a vertex feature will depend on efficient parallel graph kernels. Another example of a kernel-feature pair is breadth-first search (BFS) and the eccentricity of a vertex, which is the maximum distance from v to any vertex in the connected component of v. The eccentricity of v can be computed by measuring the height of the BFS tree rooted at v.

Vertex features are mathematically useful ways to summarize the topological information contained in the edge set. Each feature compresses the information in the graph; however by compressing it differently, an ensemble of vertex features can extract higher-level features and properties from the graph. Each feature also defines a sense in which two vertices are similar. For example, two vertices with the same degree have the same neighborhood size, and two vertices with the same clustering coefficient have the same local triangle structure.

One implication of this framework for graph analysis is that the computation of these vertex features will produce a large amount of extracted data from the graph. As shown in Fig. 1, the data for each statistic can be stored as an $V \times T$ array, which is indexed by vertex set V and time-steps $T = t_1, t_2, \ldots, t_n$. These dense matrices are amenable to parallel processing using techniques from high performance linear algebra. In Section 4.2 we apply both Nonnegative Matrix Factorization (NMF) and Singular Value Decomposition (SVD) in order to infer the temporal relationships between the vertices. Once we have created these dense matrices of features, we can apply large scale data analysis techniques in order to gain insight from the graph in motion, mainly to study the rise and fall of influential nodes over time. Another advantage of the vertex feature approach is the ability to visualize the large temporal graph. Typical visualizations attempt to show the nodes and connections on the plane capturing the spatial relationship of the nodes. Using vertex features, one can see the relationship between the behavior of the vertices without bombarding the user with all of the edges. This is demonstrated along with our observations in Section 5.

3.2. Stinger

STINGER is a high performance graph analysis package that runs on large shared memory parallel computers. The key data structure is a blocked adjacency list that allows for efficient insertion, modification and deletion of edges. [3] The

[2] This is distinct from a kernel function that compares the similarity of two graphs.

[3] See <http://www.stingergraph.com/> for code and data.

paradigm for an algorithm using STINGER is to perform initial work using a static graph that is possibly empty along with parameters to the algorithm. Then as edges are inserted from a stream, the algorithm (kernel) maintains a data structure on the graph as changing edges necessitate changes to the data structure. These edges are received as batches which allow for parallelism. After each update a vertex feature is computed for each vertex in parallel and this is transmitted to another process that is awaiting the values of the vertex feature. The STINGER package allows for coordination between kernels so that multiple vertex features can be computed simultaneously without duplicating the effort and memory requirements of updating and storing the graph. This coordination is handled by running a main server which transmits the new edge updates to each kernel and then synchronizes the stages of computation.

Each algorithm produces a vector of length $|V|$ that is stored for analysis. The computation after each batch considers all edges in the graph up to the current time. As the graph grows, the memory will eventually become exhausted, requiring edges to be deleted before new edge insertions can take place. We do not yet consider this scenario, but propose a framework by which we can analyze the graph in motion.

An evaluation of the performance of the STINGER platform and various vertex features is presented in [10]. The time for various algorithms is measured by running them on an Intel Xeon E5–2670 with 16 hyperthreaded cores and 64 GiB of main memory. A fully operational server which is concurrently performing the data ingest and each feature computation is used to give an approximation of a realistic workload in operation. The features computed are degree velocity, pagerank, and approximate betweenness centrality, where degree velocity is the change in degree for each vertex. The updates are performed in batches of 5000 for the purpose of exposing parallelism, because inserting a single edge into an adjacency list does not present an opportunity for parallel computation. Large batches also amortize the overhead of communicating the edges between processes. Since the computation of different kernels takes different amounts of time per batch and STINGER preserves the temporal ordering of the edge stream, we cannot allow faster algorithms to run ahead of the slower algorithms. This produces an overall system that takes as long as the slowest algorithm.

A total of 53 batches of 5000 edges were ingested from the Hurricane Sandy Twitter dataset. The average time required to insert and update 5000 edges in the STINGER data structure was 2.89 ms with a median of 2.85 ms. This yields an update rate of 1.73 million updates per second. The degree velocity kernel had an average update time of 24.2 ms per 5000 edges and a median update time of 24.4 ms, and an update rate of 207,000 updates per second. The PageRank kernel had an average update time of 132 ms per 5000 edges and a median update time of 126 ms, with a capacity to process 38,000 updates per second. The betweenness centrality kernel had an average update time of 214 ms per 5000 edges and a median update time of 193 ms, processing 23,400 updates per second. As you can see, the more complex information takes longer to extract, with the degree velocity closer to the raw update time than pagerank and betweenness centrality. Global information about the graph is more expensive than local information, but this additional computation provides deeper insight into the vertex behavior.

3.3. Block Principal Pivoting Algorithm for NMF

After computing vertex features using STINGER, we will apply NMF in order to make inferences about vertex and graph behavior. Here we discuss the NMF problem for general matrices $\mathbf{F}$. Given a non-negative matrix $\mathbf{F} \in \mathbb{R}_+^{m\times n}$, the problem of Non-negative Matrix Factorization(NMF) is to find two matrices $\mathbf{W} \in \mathbb{R}_+^{m\times k}$ and $\mathbf{H} \in \mathbb{R}_+^{k\times n}$ such that $\mathbf{F} \approx \mathbf{WH}$. Formally the NMF problem can be defined as

$$\mathbf{W}^*, \mathbf{H}^* = \arg\min_{\mathbf{W},\mathbf{H}} \|\mathbf{F} - \mathbf{WH}\|_F^2 \quad \text{s.t., } \mathbf{W} \geqslant 0; \quad \mathbf{H} \geqslant 0; \tag{1}$$

The NMF problem is non-convex to solve $\mathbf{W}$ and $\mathbf{H}$ together. However, if we assume one of them is given, solving the other is a convex problem. Hence, we alternatively solve two sub problems of finding $\mathbf{W}$ and $\mathbf{H}$ untila stopping criteria.

$$\mathbf{H} \leftarrow \arg\min_{\mathbf{H}\geq 0} \|\mathbf{WH} - \mathbf{F}\|_F^2 \text{ and } \mathbf{W} \leftarrow \arg\min_{\mathbf{W}\geq 0} \|\mathbf{H}^\mathsf{T}\mathbf{W}^\mathsf{T} - \mathbf{F}^\mathsf{T}\|_F^2. \tag{2}$$

By taking transposes, we see that the algorithm for finding $\mathbf{W}$ is the same as the algorithm for finding $\mathbf{H}$, thus we focus our attention on solving for $\mathbf{H}$. The columns can be partitioned into independent blocks and each block can be solved for with a concurrent NNLS with multiple right hand sides. We leverage the fact that if $\mathcal{I}$ is a partition of the index set, we can expand the Frobenius norm as shown in Eq. 3.

$$\|\mathbf{WH} - \mathbf{F}\|_F^2 = \sum_{I\in\mathcal{I}} \sum_{i\in I} \|\mathbf{W}\mathbf{h}_i - \mathbf{f}_i\|_2^2 \tag{3}$$

There are different algorithms for solving the above NMF problem. Also there are many variants depending on the characteristics of the input matrix such as symmetric [26], bounded [23] etc. For a general non-negative input matrix, the most common algorithms are multiplicative update [28], Hierarchical Alternative Least Squares (HALS) [6] and Block Principal Pivoting [25]. Kim, He and Park [24], present a detailed comparison and the properties of these algorithms using Block Coordinate Framework. For this paper, we are using BPP as it is the fastest and scalable NMF algorithm.

$$\sum_{I\in\mathcal{I}} \|\mathbf{W}\mathbf{H}_I - \mathbf{F}_I\|_F^2 \quad ; \text{where,} \quad \mathbf{H}_I \in \mathbb{R}_+^{k\times|I|} \tag{4}$$

We can decompose the Frobenius norm into a sum over columns and consider only a subset of the columns, which gives Eq. (4). Minimization of the above expression is a non-negative least squares (NNLS) problem with multiple right hand sides. In general the NNLS problem with multiple right hand sides is

$$\min_{\mathbf{X}\geqslant\mathbf{0}}\|\mathbf{CX}-\mathbf{B}\|_F^2 \tag{5}$$

The Block Principal Pivoting (BPP) algorithm listed as Algorithm 1 for the above problem is discussed by Kim and Park [25]. We are using this algorithm because it has scalable performance as demonstrated in 6. Here we briefly explain the algorithm which is an iterative algorithm inspired by the active set method. If we knew which indices correspond to nonzero values in the optimal solution, then computing the optimal solution is an unconstrained least squares problem on these indices. Call the set of indices i such that $x_i = 0$ the active set and the remaining indices the passive set. The BPP algorithm works to find this active set and passive set. Since the above problem is convex, the correct partition of the optimal solution will satisfy the Karush–Kuhn–Tucker (KKT) condition. The BPP algorithm greedily swaps indices between the active and passive sets until finding a partition that satisfies the KKT condition. In the partition of the optimal solution, the values for indices that belong to the active set will be zero. The values of the indices that belong to the passive set are determined by solving the least squares problem using normal equation [4] restricted to the passive set.

We have all the necessary building blocks to explain our parallel multicore NMF algorithm using ANLS-BPP. Broadly the algorithm has two major components. (a) Given $\mathbf{W}\geqslant\mathbf{0}, \mathbf{F}\geqslant\mathbf{0}$, find a non-negative $\mathbf{H}\geqslant\mathbf{0}$ and then given $\mathbf{F}$ and $\mathbf{H}$, find a $\mathbf{W}$, alternating until the stopping criteria is satisfied. (b) Partitioning columns of $\mathbf{H}$ and $\mathbf{W}$ callingANLS-BPP with each partition.

Algorithm 1: Multicore NMF Algorithm

input : Matrix $\mathbf{F}\in\mathbb{R}_+^{m\times n}$, target rank k
output: Matrices $\mathbf{W}\in\mathbb{R}_+^{m\times k}$, $\mathbf{H}\in\mathbb{R}_+^{k\times n}$

Initialize $\mathbf{W}$ *as nonnegative random matrix* ;
while *stopping criteria not met* **do**
  `// Find matrix H given F,W`
  $\mathbf{H}\leftarrow\text{findOptimal}(\mathbf{F},\mathbf{W})$;
  `// Find matrix W given F,H`
  `// Transpose the returned matrix`
  $\mathbf{W}\leftarrow\text{findOptimal}(\mathbf{F}^\intercal,\mathbf{H}^\intercal)^\intercal$;
end
function `findOptimal`$(\mathbf{F}\in\mathbb{R}_+^{m\times n},\mathbf{W}\in\mathbb{R}_+^{m\times k})$
  $\mathcal{I}\leftarrow\text{Partition}(1,\cdots,n)$;
  for $I\in\mathcal{I}$ **do in parallel**
    `// Assign the returned matrix from BPPNNLS to the index` I `of` $\mathbf{H}$
    `// BPPNNLS with multiple right hand sides from [25]`
    `// According to equation (5),` $\mathbf{C}=\mathbf{W}$ `and` $\mathbf{B}=\mathbf{F}_I$
    $\mathbf{H}_I\leftarrow\text{bppnnls}(\mathbf{W},\mathbf{F}_I)$;
  end

3.3.1. NMF parallelism theory

The *ANLS* − *BPP* routine in Algorithm 1 is an iterative method. It requires the computation of few matrix matrix products once and a least square for each iteration of the method. The matrix matrix multiplication or level-3 BLAS cost is $O(mkn+k^2m)$ and the time spent over all iterations is $O(k^4(m+n))$. The upper bound comes from the fact that the active set method can take at most k iterations in order to find a solution of length k and we are solving for m NNLS vectors for $\mathbf{W}$ and n NNLS vectors for $\mathbf{H}$. Each iteration of the active set method requires at most a Cholesky decomposition for $k\times k$ matrix and a two triangular solves involving $k\times k$ matrix. Since the parallelization is over the number of solution vectors, we can see a parallel run time on p processors that is $O(k^4m/p)$ when solving for $\mathbf{W}$ and $O(k^4n/p)$ when solving for $\mathbf{H}$ where p is limited to m or n respectively. The asymptotic cost to compute the product of a $m\times k$ matrix with an $n\times k$ matrix is well known as $O(mkn/p)$ in shared memory parallel computers. This indicates that for rank k decompositions where k^3 exceeds either m or n, the limiting step in the computation is the k least squares step for the larger of $\mathbf{W}$ or $\mathbf{H}$. In detail assume $m>k^3>n$, then $mkn>nk^4$ so matrix multiply exceeds the cost of the k least squares solve for $\mathbf{H}$. However, $mkn<mk^4$ implying that the cost of least squares for $\mathbf{W}$ exceeds the matrix multiplication cost. In our case there are more vertices than time-steps and so the k least squares is the dominant cost when k^3 exceeds n. One implication of this analysis is that the chosen number of communities effects not only the overall runtime of the algorithm but also which step is the dominant cost. We study these runtime considerations empirically in Section 6.

3.4. Graph similarity metrics

One task when analyzing a dynamic graph, is to determine how much the graph has changed over a period of time. We will use several measures to address this task and draw some conclusions about their relative merit. The simplest measure

[4] The solution to least squares problem $||\mathbf{Ax}-\mathbf{b}||_2^2$ is obtained by solving the system of linear equations for $\mathbf{A}^\intercal\mathbf{Ax}=\mathbf{A}^\intercal\mathbf{b}$

determines the similarity without looking at which edges we insert but only the number of edges inserted and the size of the graph. We define the relative impact of a batch of insertions as the number of edges in the batch divided by the average size of the graph during the insertions. Letting the $E_i = bi$ be the number of edges in the graph at time i, which corresponds to a batch size of b edges, we obtain Eq. 6 for the relative impact of batches i to $i+s$.

$$RI(i, i+s) = \frac{2bs}{bi + b(i+s)} = \frac{s}{i + s/2} = \left(\frac{i}{s} + \frac{1}{2}\right)^{-1} \tag{6}$$

The relative impact allows us to reason a priori about what the behavior of a measure of graph similarity over time. If we fix a gap or delay of size s then the impact of each batch decreases over time. Also as the delay s increases, the relative impact increases. The similarity of a dynamic graph to future instances should behave as the inverse of the relative impact of those edge insertions. Thus similarity should weaken as the gap size increases and similarity should strengthen as the size of the graph increases.

Additionally, we can define a measure of similarity for any set of vertex features. Let $\mathbf{F} \in \mathbb{R}^{m \times n}$ be a feature matrix, where each row represents a vertex and each column represents a time stamp, $(\mathbf{F}^\intercal \mathbf{F})_{ij}$ represents the similarity of the graph at time i to the graph at time j. These similarities are the inner product (cosine) similarities one gets by representing the graph as a point in n dimensional space where the vth coordinate is the value of the vertex feature for vertex v. These similarities are specialized to particular vertex features and their nature is defined by the behavior captured by the chosen vertex features.

Using matrix factorization $\mathbf{F} \approx \mathbf{WH}$ we can get another similarity measure defined as $\mathbf{H}^\intercal\mathbf{H}$. This similarity matrix takes account of the relationships between the vertex feature at different vertices. The nature of the factorization determines the behavior of this similarity. For example, the Singular Value Decomposition (SVD) for a given rank $\mathbf{F} \approx \mathbf{U\Sigma V}^\intercal$ gives two orthogonal matrices $\mathbf{U}, \mathbf{V}$ and a nonnegative diagonal matrix $\mathbf{\Sigma}$ that best approximate $\mathbf{F}$ in the two norm among rank k matrices. We can compute a similarity based on this factorization as $\mathbf{F}^\intercal\mathbf{F} \approx \mathbf{V\Sigma}^\intercal\mathbf{\Sigma V}^\intercal$. By definition of the SVD, this similarity will contain a smooth approximation of the similarity and will account for the interrelation of the vertices. The orthogonal vectors found by the SVD are those that best represent the variance in the data. Each vector well approximates the variance of the entire data set. According to Kuang, Yun, and Park [26] NMF is a form of clustering, where $\mathbf{W} \in \mathbb{R}^{m \times k}$ is a set of basis vectors such that each basis vector is a representative of a cluster, and $\mathbf{H} \in \mathbb{R}^{k \times n}$ represents the distribution of every data point over these k clusters. Hence, $\mathbf{H}$ identifies clusters in time. Thus $\mathbf{H}^\intercal\mathbf{H}$ will represent the graph similarity accounting for the clustered structure of the vertex set. In contrast to the SVD, each representative of a cluster is predominantly determined by the members specific to that cluster. We observe the differences between the SVD and NMF in Section 5. These similarity measures are useful ways to compare each snapshot of the graph to previous and future snapshots.

4. Temporal analysis

In order to explain our temporal analysis and construction of useful vertex features, we introduce a running example using Twitter data. Specifically, we assembled a corpus around a single event maximizing the likelihood of on-topic interaction and interesting structural features. At the time, there was great concern about the rapid development of Hurricane Sandy. Weather prediction gave more than one week of advanced notice and enabled us to build a tool chain to observe and monitor information regarding the storm on a social network from before the hurricane made landfall through the first weeks of the recovery effort. For diversity we also demonstrate these techniques with applications to CAIDA data that was collected passively on internet traffic.

In order to focus our capture around the hurricane, we selected a set of hashtags (user-created metadata identifying a particular topic embedded within an individual Twitter post) that we identified as relevant to the hurricane. These were *#hurricanesandy, #zonea, #frankenstorm, #eastcoast, #hurricane,* and *#sandy*. Zone A is the evacuation zone of New York City most vulnerable to flooding.

The data set includes 1.4 million public Twitter posts starting from the day before the storm made landfall as an edge list. This edge list includes any mentions of one user by another as well as cases where a user "retweeted" or reposted another user's post, because retweets mention the author of the original Tweet similar to a citation. The dataset included over 1,238,109 mentions from 662,575 unique users. We construct a graph from this stream in which each username is represented as a vertex. The file contains a set of tuples containing two usernames which are used to create the edges in the graph. The temporal ordering of the mentions is maintained through the processing tool chain resulting in a temporal stream of mention events encoded as graph edges. The edge stream is divided into batches of 10,000 edge insertions for our analysis. As each batch is applied to the graph, we compute the betweenness centrality, local clustering coefficient, number of closed triangles, and degree for each vertex in the graph.

4.1. Temporal features

In order for this framework of connecting graph algorithms to machine learning algorithms to be applied, one must first choose a set of vertex features to compute. We discuss here how one can study a direct vertex feature, as well as temporal features that can be derived from the direct features.

4.1.1. Betweenness centrality

Centrality metrics on static graphs provide an algorithmic way to measure the relative importance of a vertex with respect to information flow through the graph. Higher centrality values generally indicate greater importance or influence. Betweenness centrality [16] is a specific metric that is based on the fraction of shortest paths on which each vertex lies. The computational cost of calculating exact betweenness centrality can be prohibitive; however, approximating betweenness centrality is tractable and produces relatively accurate values for the highest centrality vertices [12,3]. It is known that betweenness centrality follows a heavy tail distribution. In order to account for the heavy tail, we examine the logarithm of the betweenness centrality score for each vertex. Since many vertices have zero or near zero betweenness centrality we add one before taking the logarithm and discard vertices with zero centrality.

For the Twitter data set after inserting 98 batches of edges we find that the right half of the distribution of logarithm of betweenness centrality can be modeled as an exponential distribution. The cumulative distribution function (*CDF*) is well approximated by $F(x) = 1 - \exp -\lambda(x - x_0)$ where the maximum likelihood estimates for the location and rate parameters are $x_0 = 5.715$ $\lambda = 1.205$ respectively. Since betweenness centrality estimates are more accurate for the high centrality vertices [17], we focus our analysis on the vertices whose centrality is larger than the median.

Fig. 2a shows both the empirical *CDF* and the modeled *CDF* for the log(betweenness centrality). It is apparent in the figure that the exponential distribution is a good fit for the right tail. We can use the *CDF* to assign a probability to each vertex, and these probabilities can be consumed by an ensemble method for a prediction task. This will allow traditional machine learning and statistical techniques to be combined with high performance graph algorithms while maintaining the ability to reason in a theoretically sound way. Such distributional analysis connects the analysis of graph topology to the well studied fields of parameter estimation and nonparametric density estimation. Understanding the distribution of these features allows us to choose transformations of the data that will be helpful in follow-on analysis.

In Fig. 3a, we trace the value of betweenness centrality for a selection of vertices over time. In the sociological literature, this corresponds to a longitudinal study. It is clear that there is a significant amount of activity for each vertex. Such a longitudinal study of vertices can be performed for any vertex feature that can be devised for graphs.

4.1.2. Finite differences

Tracking the derivatives of a feature can provide insight into changes that are occurring in a graph in real time. Eq. 7 defines the (discrete) derivative of a vertex feature for a vertex v at time t using the following equation where $b(t)$ is the number of edges inserted in batch t. Note that the derivative of a vertex feature is also a vertex feature.

$$\frac{df}{dt}(v,t) = \frac{f(v,t+1) - f(v,t-1)}{b(t) + b(t+1)} \tag{7}$$

When concerned about maximizing the number of edge updates that can be processed per second, fixing a large batch size is appropriate. However when attempting to minimize the latency between an edge update and the corresponding update to vertex features, the batch size might vary to compensate for fluctuations in activity on the network. Dividing by the number of edges per batch accounts for these fluctuations. For numerical or visualization purposes one can scale the derivative by a constant.

The data can be examined in a cross section by examining the distribution of the derivatives at a fixed point in time. Fig. 2b shows the *CDF* of $\frac{d}{dt} \mid log(BC) \mid (v)$ at batch 98 on a log scale. Where the vertices are grouped by the sign of their derivative. The separation of the two curves show that the distribution of increases in logarithm of betweenness centrality is different than the distribution of decreases in logarithm of betweenness centrality. By counting the number of vertices of each group, we determined that the most vertices decrease in betweenness centrality in this batch.

For example, Fig. 3b shows the derivative of logarithm of betweenness centrality. These traces indicate that changes in the betweenness centrality of a vertex are larger and more volatile at the beginning of the observation and decrease in magnitude over time. The reason for taking logs before differentiation is that it effectively normalizes the derivative by the value for that vertex.

Because the temporal and topological information in the graph is summarized using real numbers, we can apply techniques that have been developed for studying measurements of scientific systems to graphs. Since the derivative of a vertex feature is another vertex feature, these derivatives can be analyzed in a similar fashion. By estimating the distribution of $\frac{df}{dt}$ for any feature we can convert the temporal information encoded in the graph into a probability for each vertex.

4.2. Temporal graph analysis using NMF

Here we reiterate the connections among NMF, vertex feature extraction and graph analysis. Our objective is to study the behavioral changes of vertices and the graph over time. Towards this end, we construct vertex features using highly scalable infrastructure such as STINGER. We study the vertex changes over time using the exploratory data analysis techniques explained in the previous section. Given the vertex features and the temporal features, in this section we explain our novel method to understand behavioral clusters and temporal changes in the graph using NMF.

Identifying and constructing explicit features of graphs that are good for understanding temporal changes is difficult. In order to avoid constructing explicit features, we construct implicit (latent) features through the low rank approximation of

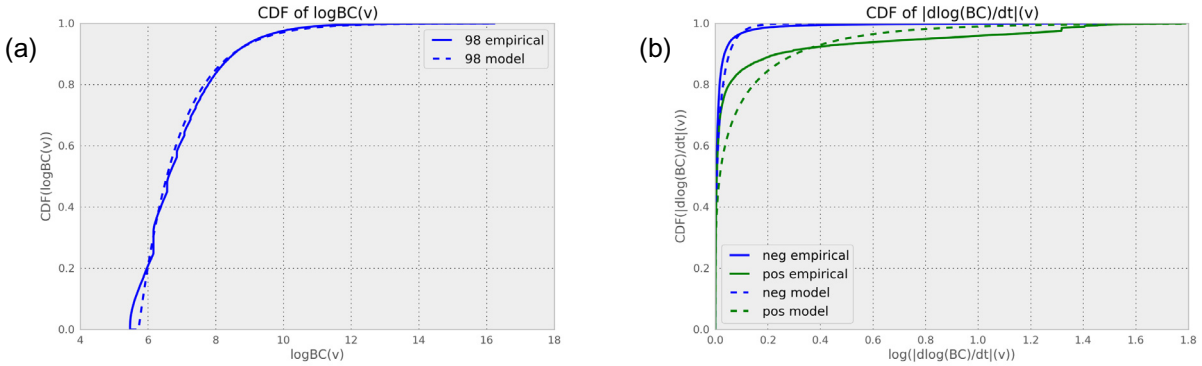


Fig. 2. (a) The cumulative distribution function for logarithm of betweenness centrality empirical (solid) and exponential best fit (dashed) (b) CDF of the derivative evaluated at time 98. Vertices with increasing Betweenness centrality are separated to show the difference in distribution.

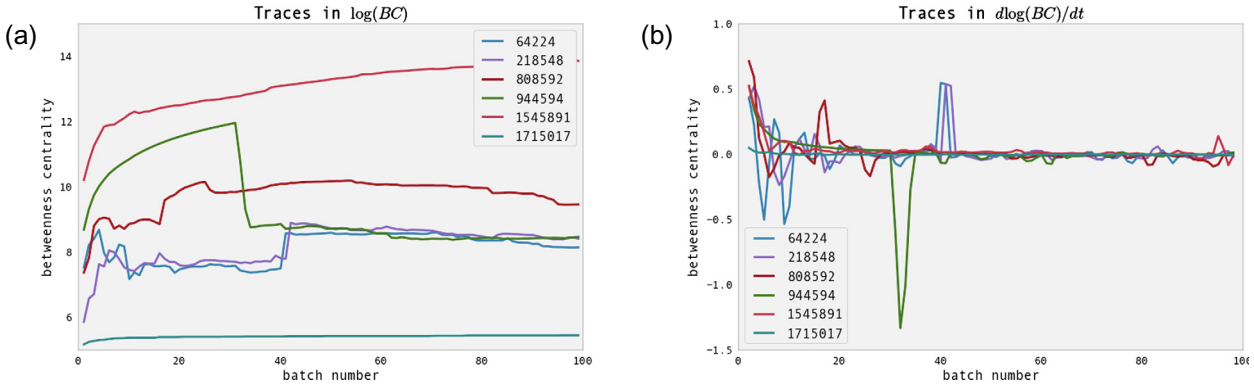


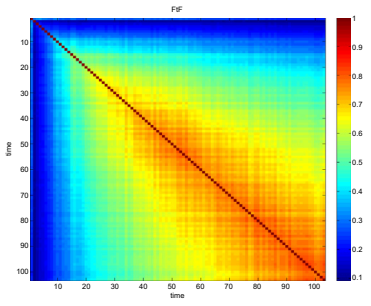
Fig. 3. (a) Traces of betweenness centrality value for selected vertices over time. (b) The derivative of the logarithm of betweenness centrality values for selected vertices.

the original feature matrix. These implicit features capture the structure of the vertex features extracted from the graph. Given the temporal edge stream containing N vertices, we form a matrix $\mathbf{F} \in \mathbb{R}_{+}^{m \times n}$ matrix where n is the number of time-steps and $m = dV$ is the number of vertex and temporal features times the number of vertices, as explained in previous section. Given this matrix $\mathbf{F}$, we build graph features of dimension k over n time-steps, such that these latent features are good representation for the graph as a whole. It is important to appreciate the difference between vertex features and latent features. Vertex features are constructed from the graph structure such as between centrality, pagerank, and clustering coefficient, whereas latent features are implicitly defined from the values of these vertex features.

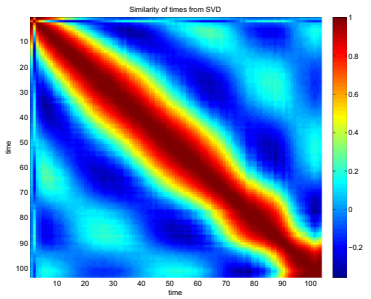
In our scenario, we use NMF to separate the input matrix $\mathbf{F} \in \mathbb{R}_{+}^{m \times n}$, which relates the vertex features to the time-steps, into $\mathbf{W} \in \mathbb{R}_{+}^{m \times k}$, which relates the vertex features to the latent features, and $\mathbf{H} \in \mathbb{R}_{+}^{k \times n}$, which relates the latent features to the time-steps. This process discovers k latent features that mediate the interactions between vertices and time-steps. The matrix $\mathbf{W}$ reveals a clustering on vertices, and the matrix $\mathbf{H}$ provides a representation for studying the temporal changes in the graph as a whole. Since a direct clustering of $\mathbf{F}$ would not produce both pieces of information concurrently, we choose a low rank approximation approach. In the next section we show a case study from a real world graph and demonstrate the usefulness of $\mathbf{W}, \mathbf{H}$ for understanding the temporal behavior in the dynamic graph.

5. Case study – CAIDA dataset

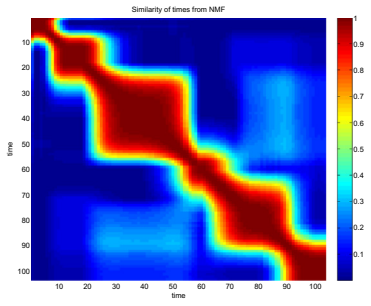
In this section, we present the observation and analysis of temporal clusters generated by NMF. As illustrated in Fig. 1, we are using the explicit vertex features matrix as input to the NMF. The output of NMF gives us the temporal clusters of nodes that aids us in visualizing the rise and fall of influential nodes. In order to study the utility of these vertex feature matrices, we conduct experiments on IP network data. The STINGER library is used to extract the betweenness centrality and pagerank for each vertex at each time-step. Because this is a bipartite network we do not find any triangles. The CAIDA passive traces form an IP network where vertices are host IP addresses and an edge means that a packet was sent from one host to another. This graph has timestamped edges extracted from the packet capture (`pcap`) data. CAIDA releases this data in an



(a) $\mathbf{F}^\intercal\mathbf{F}$ does not account for any dependence between the vertices or between the derived features. This shows only the large scale trend.



(b) The SVD accounts for intervertex and interfeature dependence in a smooth manner.



(c) The $\mathbf{H}$ factor from NMF gives an approximately block diagonal structure. This shows clusters in time.

Fig. 4. The temporal similarity of betweenness centrality structure between all pairs of time-steps, in internet traces data.

anonymized form under an acceptable use policy and it can be obtained from them [35]. Edges are processed as undirected for the purpose of graph kernels. We examine logarithm of betweenness centrality, its squared discrete derivative, and the exponentially weighted moving average and exponentially weighted moving standard deviation. These derived statistics capture the temporal nature by finite differences and the distributional aspects by capturing the center and spread of the distribution of recently observed data. These features are arranged as a matrix $\mathbf{F} \in \mathbb{R}_+^{m\times n}$. Since we perform only edge insertions we use exponentially weighted moving averages to discount the historical data. If more augmenting features are found to be interesting, they can be computed without impacting the behavior of the graph algorithmic code.

In this context, the matrix $\mathbf{F}^\intercal\mathbf{F}$ (normalized so that $(\mathbf{F}^\intercal\mathbf{F})_{i,i} = 1$) gives similarity of the graph over time. The nature of the graph similarity is determined by the choice of feature. If one is interested in the influence structure of the graph, then influence metrics such as betweenness centrality and pagerank can be used. The i,jth entry of this matrix is the similarity between the graph at time t_i to time t_j. When we examine this matrix in Fig. 4a all we see is the large scale trend. By factorizing this matrix into low rank factors, we reveal a more refined picture of the dynamics. Fig. 4b shows the similarity after using the Singular Value Decomposition (SVD) $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$ to account for the intervertex dependencies.

The approximate block structure of $\mathbf{H}^\intercal\mathbf{H}$ indicates a clustering on the rows and columns. The NMF has determined that certain time-steps belong together. Because we are inserting edges into the graph, we are not surprised that the clusters are identified as contiguous subsets of the time domain. By comparing these similarity matrices to the formula for relative impact given in Eq. 6, we see that all three of these similarity functions agree with the dependence on s. As the gap between two observations i and $i+s$ increases, the similarity of the graph at those two times decreases. In the case of the basic similarity that does not account for vertex interactions, we see only this trend effect. The SVD accounts for the vertex interactions and gives a more specific similarity. However, the real difference occurs when we examine the cluster structure of the data. The NMF similarity $\mathbf{H}^\intercal\mathbf{H}$ gives a discrete sense of similarity. If two time-steps are in the same phase (cluster of time-steps), then the similarity is very high, and if they are in different clusters the similarity is very low. This validates the claim that NMF produces a segmentation of the edge stream into phases of activity. Pivoting our attention to $\mathbf{W}$, the left hand factor, we discover the clusters of vertices. The index of the largest element in each row of $\mathbf{W}$ indicates to which cluster that vertex belongs. Returning to the longitudinal study of vertices and grouping the vertices into their clusters, Fig. 5 shows the plot of logarithm of betweenness centrality for each vertex over time, which reveals a clear pattern. Most elements of each cluster peak in betweenness centrality around the same time. The timing and duration of these peaks correspond to the diagonal blocks of $\mathbf{H}^\intercal\mathbf{H}$. We have clustered vertices into groups that rise and fall in influence together. The similarly matrix $\mathbf{V}\mathbf{\Sigma}^\intercal\mathbf{\Sigma}\mathbf{V}^\intercal$, is smoother as every data point is projected onto the basis $\mathbf{U}$ representing the entirety of the data. Both the SVD and NMF approximations account for the interdependence of vertices and features, thus revealing more information than $\mathbf{F}^\intercal\mathbf{F}$. In the next section we present the scalability experiments of the NMF algorithm.

6. Performance experiments

In order to validate the application of NMF to graph analysis, we give empirical evidence that the parallel BPP algorithm presented in Section 3.3 is scalable. The experiments are conducted on a dual socket Intel Xeon CPU E5–2620 machine clocked at 2.00 GHz. Each socket has 6 cores along with hyperthreading, hence there are 12 cores in total and 24 threads of execution. Our input matrix is a dense matrix of size 2840256x103, where 103 represents number of time-steps and 2840256 are 8 different features observed over each of 355032 vertices. One of the parameters for BPP is the rank k which determines the number of clusters and for practical applications, k is chosen on the order of 10's. Since the rank of the matrix is 103 (the number of independent columns) we show experiments where $k = 10, 25, 50$. For this experimentation, our code

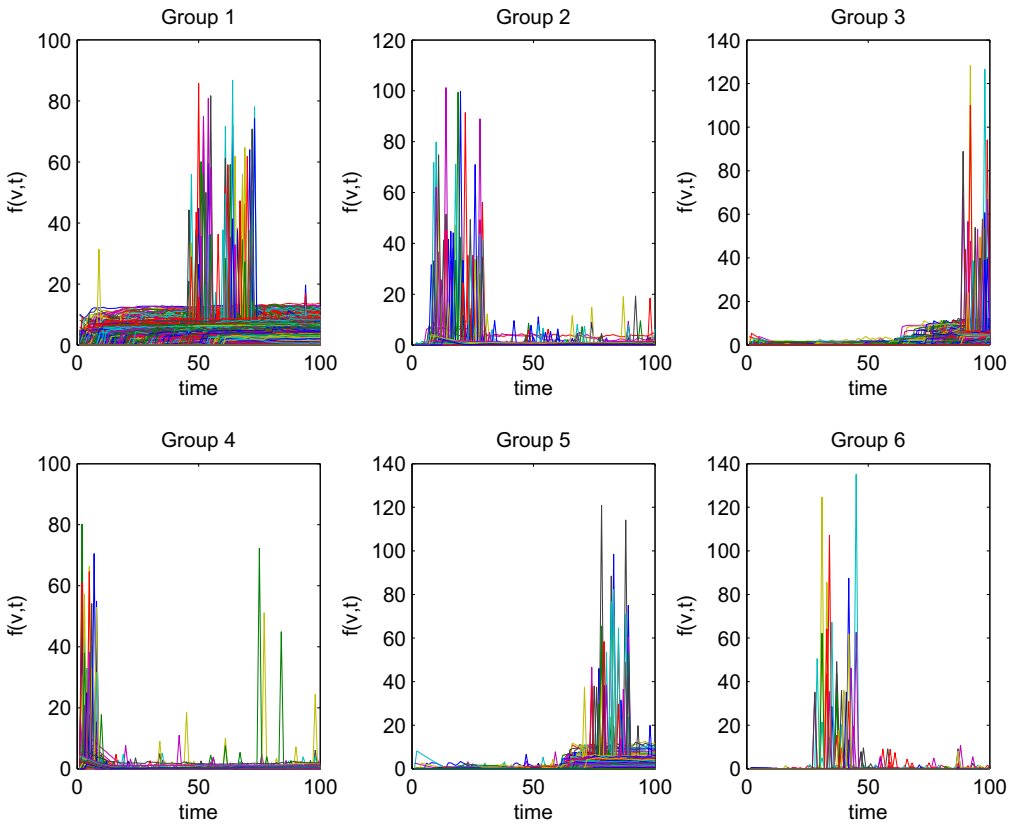
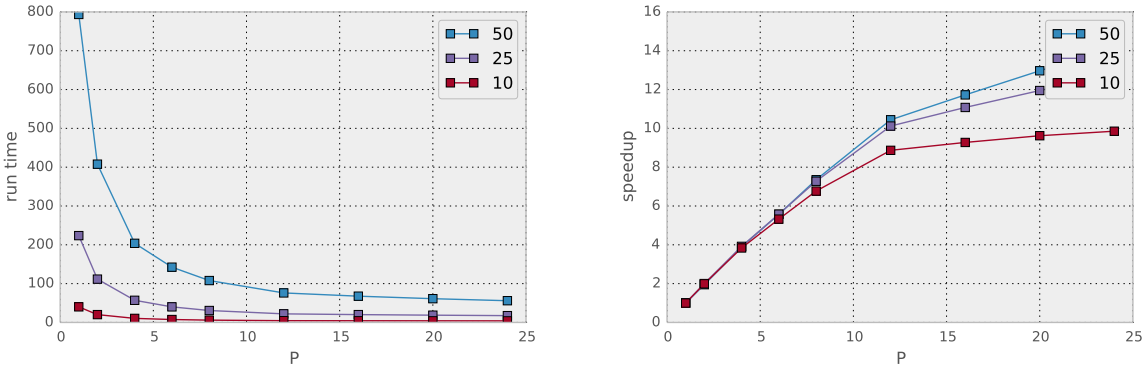


Fig. 5. Longitudinal plot of a sample of vertices from each cluster identified by NMF on the betweenness centrality derived feature matrix. Each subplot represents a group of vertices that peak at the same time. The logarithm of betweenness centrality is shown on the ordinate.

uses the Intel MKL implementation of BLAS and LAPACK. Since our machine has 12 physical cores but can support 24 threads with hyperthreading, we choose the number of threads as 1, 2, 4, 6, 8, 12, and 24 so that we also study the performance when the number of threads is more than number of physical cores. In the Fig. 6, the number of cores is presented on the x-axis and the running time (Fig. 6a) and speedup factor (Fig. 6b) is shown on the y-axis. From these graphs we observe linear scaling up to the number of physical cores on the system. When hyperthreading is used, the speedup is no longer linear, as we achieve a speedup of approximately 10 on 12 cores and a speedup of approximately 12 on 24 cores. From these experiments we also infer that the algorithm achieves less parallel efficiency when forming a very low rank approximation of the matrix. Since the work per core is $O(k^4(m+n)/p)$ we expect more parallel efficiency for larger values of k than for very small values of k.



(a) NMF Runtime (b) NMF Speedup

Fig. 6. NMF–BPP Scalability experiments on a dense matrix of size 2,840,256 × 103.

According to the BPP algorithm, when we find **H**, the majority of the time is taken towards computing the matrix multiplications $\mathbf{W}^T\mathbf{F}$ and $\mathbf{W}^T\mathbf{W}$, which surpasses the time needed for computing the small matrix **H** of size 103xk. Where as, in the case of computing **W**, the computational effort for the matrix multiplications $\mathbf{H}^T\mathbf{H}$ and $\mathbf{H}\mathbf{F}^T$ was smaller than for the iteration to find **W**. Thus the effort for finding every vector $\mathbf{w}_i$ is very scalable with multiple threads and this advantage is more pronounced for $k = 50$. The observed linear scaling with better efficiency for larger values of k confirms the behavior predicted by theory. One implication of this parallel performance analysis is that the partitioning of matrices for distributed memory processing must account for the asymmetry between the left and right factors.

7. Conclusions and future work

We contribute a novel framework for making inferences about dynamic vertex behavior based on streaming graph computations. This framework provides a new method for detecting clusters of vertices based on temporal behavior. The inferred temporal behavior is interpretable, easy to visualize, and found without explicitly searching for a predefined pattern. We developed a successful feature based pipeline for dynamic graph streams and applied NMF to these features. The computed low rank approximation is useful for several graph analysis tasks including temporal similarity at the graph level, vertex clustering and graph temporal segmentation. We contribute a theoretical and empirical evaluation of a new parallel algorithm for NMF based clustering, which scales up to 12 cores and exhibits good performance up to 50 clusters, even on graphs with large vertex sets. These techniques yield an algorithm which is linear in the number of vertices m and time-steps n when the number of temporal behavior clusters is $O((m+n)^{1/4})$. A case study analyzing CAIDA internet traces provides understanding into the behavior of the ubiquitous internet connection graph. This feature based pipeline takes advantage of longitudinal studies of vertex behavior over time to inform the generation of derived features. A low rank approximation to the matrix of features produces clusters of vertices that rise and fall in influence together. The clusters of vertices correlate with sharp boundaries in time which form phases of the network's evolution. Our approach loosely couples parallel graph algorithms with machine learning algorithms, which allows the application of both systems effectively.

Future work includes tensor analysis on these temporal features. By restricting to methods using matrices instead of tensors, we loose the coupled relationship between two features evaluated on the same vertex. Future work with tensors is expected to recover this relationship. By leveraging a diverse collection of matrix factorizations, one can extract distinct insights into the vertex and graph behavior from the same vertex features. Determination of the most useful features is still an open subject. The ANLS-BPP algorithm uses nonnegative least squares with multiple right hand sides where the grouping of right hand sides is currently chosen for simplicity of implementation. In a more refined future implementation, an optimal grouping of right hand sides could be used.

Acknowledgments

This work was partially supported by the American Society for Engineering Education (ASEE) National Defense Science and Engineering Graduate (NDSEG) Fellowship as well as the National Science Foundation (NSF) Grants IIS-1348152 and ACI-1338745. This work was also partially supported by the Defense Advanced Research Projects Agency (DARPA) XDATA program Grant FA8750–12-2–0309. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ASEE, NSF or DARPA.

References

[1] K.J. Ahn, S. Guha, A. McGregor, Graph sketches: sparsification, spanners, and subgraphs, in: Proceedings of the 31st Symposium on Principles of Database Systems (PODS2012), ACM, 2012, pp. 5–14.
[2] L. Akoglu, M. McGlohon, C. Faloutsos, Oddball: spotting anomalies in weighted graphs, in: Advances in Knowledge Discovery and Data Mining (KDD2010), Springer, 2010, pp. 410–421.
[3] D. Bader, S. Kintali, K. Madduri, M. Mihail, Approximating betweenness centrality, in: Proc. 5th Workshop on Algorithms and Models for the Web-Graph (WAW2007), Vol. 4863 of Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 134–137.
[4] E. Bakshy, J.M. Hofman, W.A. Mason, D.J. Watts, Everyone's an influencer: quantifying influence on Twitter, in: Proceedings of the fourth ACM international conference on Web search and data mining (WSDM2011), ACM, 2011, pp. 65–74.
[5] M. Cha, H. Haddadi, F. Benevenuto, P.K. Gummadi. Measuring user influence in Twitter: The million follower fallacy. in: International AAAI Conference on Web and Social Media (ICWSM2010), 10 (10–17), (2010) 30.
[6] A. Cichocki, A.-H. Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E92-A, (2009) 708–721.
[7] A. Cichocki, R. Zdunek, S. Amari, Hierarchical als algorithms for nonnegative matrix and 3D tensor factorization, Lect. Notes Comput. Sci. 4666 (2007) 169–176.
[8] A. Dainotti, A. King, K. Claffy, F. Papale, A. Pescap, Analysis of a/0 stealth scan from a botnet. in: Internet Measurement Conference (IMC2012). Nov 2012, pp. 1–14.
[9] A. Dainotti, C. Squarcella, E. Aben, K. Claffy, M. Chiesa, M. Russo, A. Pescap, Analysis of country-wide internet outages caused by censorship, IEEE/ACM Trans. Networking (2014).
[10] D. Ediger, S. Appling, E. Briscoe, R. McColl, J. Poovey, Real-time streaming intelligence: Integrating graph and nlp analytics. in: Proc. High Performace Embedded Computing Workshop (HPEC2014), 2014.
[11] D. Ediger, K. Jiang, J. Riedy, D.A. Bader, Massive streaming data analytics: A case study with clustering coefficients. in: 4th Workshop on Multithreaded Architectures and Applications (MTAAP2010). Atlanta, Georgia, Apr. 2010a.
[12] D. Ediger, K. Jiang, J. Riedy, D.A. Bader, C. Corley, R. Farber, W.N. Reynolds, Massive social network analysis: Mining Twitter for social good. in: International Conference on Parallel Processing (ICPP2010), 2010b. pp. 583–593.

[13] D. Ediger, E.J. Riedy, D.A. Bader, H. Meyerhenke, Tracking structure of streaming social networks. in: 5th Workshop on Multithreaded Architectures and Applications (MTAAP2011), May 2011.
[14] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, J. Zhang, On graph problems in a semi-streaming model, Theoret. Comput. Sci. 348 (2–3) (2005) 207–216.
[15] P. Flajolet, G.N. Martin, Probabilistic counting, in: Proceedings of the 24th Annual Symposium on Foundations of Computer Science (SFCS1983), IEEE Computer Society, Washington, DC, USA, 1983, pp. 76–82.
[16] L. Freeman, A set of measures of centrality based on betweenness, Sociometry 40 (1) (1977) 35–41.
[17] R. Geisberger, P. Sanderst, D. Schultest, Better approximation of betweenness centrality. in: Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments and the Fifth Workshop on Analytic Algorithmics and Combinatorics. Vol. 129. Society for Industrial & Applied, 2008, p. 90.
[18] K. Glasgow, C. Fink, Hashtag lifespan and social networks during the London riots, in: A. Greenberg, W. Kennedy, N. Bos (Eds.), Social computing, behavioral-cultural modeling and prediction, Lect. Notes Comput. Sci., Vol. 7812, Springer, Berlin Heidelberg, 2013, pp. 311–320.
[19] E.F. Gonzales, Y. Zhang, Accelerating the Lee-Seung algorithm for non-negative matrix factorization. Dept. Comput. & Appl. Math., Rice Univ., Houston, TX, Tech. Rep. TR-05-02, 2005.
[20] K. Henderson, T. Eliassi-Rad, C. Faloutsos, L. Akoglu, L. Li, K. Maruhashi, B.A. Prakash, H. Tong, Metric forensics: a multi-level approach for mining volatile graphs. in: Knowledge Discovery and Data Mining (KDD2010), 2010, pp. 163–172.
[21] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, L. Li, Rolx: structural role extraction & mining in large graphs, in: Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD2012), ACM, 2012, pp. 1231–1239.
[22] N.-D. Ho, P.V. Dooren, V.D. Blondel, Descent methods for nonnegative matrix factorization. CoRR abs/0801.3199, 2008.
[23] R. Kannan, M. Ishteva, H. Park, Bounded matrix low rank approximation, in: IEEE 12th International Conference on Data Mining (ICDM2012), IEEE, 2012, pp. 319–328.
[24] J. Kim, Y. He, H. Park, Algorithms for nonnegative matrix and tensor factorizations: a unified view based on block coordinate descent framework, J. Global Optim. 58 (2) (2014) 285–319.
[25] J. Kim, H. Park, Fast nonnegative matrix factorization: an active-set-like method and comparisons, SIAM J. Sci. Comput. 33 (6) (2011) 3261–3281.
[26] D. Kuang, S. Yun, H. Park, Symnmf: nonnegative low-rank approximation of a similarity matrix for graph clustering, J. Global Optim. (2014) 1–30.
[27] H. Kwak, C. Lee, H. Park, S. Moon, What is Twitter, a social network or a news media? In: 19thWorld-Wide Web (WWW2010) Conference. Raleigh, North Carolina, Apr. 2010, pp. 591–600.
[28] D.D. Lee, H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 401 (6755) (Oct. 1999) 788–791.
[29] C.J. Lin, Projected gradient methods for nonnegative matrix factorization, Neural Comput. 19 (10) (Oct. 2007) 2756–2779.
[30] M. Mendoza, B. Poblete, C. Castillo, Twitter under crisis: can we trust what we RT?, in: Proceedings of the First Workshop on Social Media Analytics (SOMA2010), ACM, New York, NY, USA, 2010, pp 71–79.
[31] P. Paatero, U. Tapper, Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values, Environmetrics 5 (2) (1994) 111–126.
[32] I. Psorakis, S. Roberts, M. Ebden, B. Sheldon, Overlapping community detection using Bayesian non-negative matrix factorization, Phys. Rev. E 83 (6) (Jun. 2011) 066114.
[33] T. Sakaki, M. Okazaki, Y. Matsuo, Earthquake shakes Twitter users: real-time event detection by social sensors, in: Proceedings of the 19th International Conference on World wide web (WWW2010), ACM, 2010, pp. 851–860.
[34] A. Signorini, A.M. Segre, P.M. Polgreen, The use of Twitter to track levels of disease activity and public concern in the u.s. during the influenza a h1n1 pandemic, PLoS ONE 6 (5) (2011) e19467.
[35] The CAIDA UCSD, 2014. passive traces – 2014. <http://www.caida.org/data/passive/passive2014dataset.xml>
[36] H. Tong, C.-Y. Lin, Non-negative residual matrix factorization: problem definition, fast solutions, and applications, Stat. Anal. Data Min. (SDM2012) 5 (1) (2012) 3–15.
[37] H. Tong, S. Papadimitriou, J. Sun, P.S. Yu, C. Faloutsos, Colibri: Fast mining of large static and dynamic graphs, in: Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD2008), ACM, 2008, pp. 686–694.
[38] F. Wang, T. Li, X. Wang, S. Zhu, C. Ding, Community discovery using nonnegative matrix factorization, Data Min. Knowl. Discovery 22 (3) (Jul. 2010) 493–521.
[39] H. Wang, M. Tang, Y. Park, C.E. Priebe, Locality statistics for anomaly detection in time series of graphs, IEEE Trans. Signal Proces. 62 (3) (2013) 703–717.
[40] J. Yang, J. Leskovec, Overlapping community detection at scale: a nonnegative matrix factorization approach. in: Proceedings of the International Conference on Web Search and Data Mining (WSDM2013), 2013, pp. 587–596.