HEigen: Spectral Analysis for Billion-Scale Graphs

U Kang, Brendan Meeder, Evangelos E. Papalexakis, and Christos Faloutsos

Abstract—Given a graph with billions of nodes and edges, how can we find patterns and anomalies? Are there nodes that participate in too many or too few triangles? Are there close-knit near-cliques? These questions are expensive to answer unless we have the first several eigenvalues and eigenvectors of the graph adjacency matrix. However, eigensolvers suffer from subtle problems (e.g., convergence) for large sparse matrices, let alone for billion-scale ones.
We address this problem with the proposed HEIGEN algorithm, which we carefully design to be accurate, efficient and able to run on the highly scalable MAPREDUCE (HADOOP) environment. This enables HEIGEN to handle matrices more than *1000*× larger than those which can be analyzed by existing algorithms. We implement HEIGEN and run it on the M45 cluster, one of the top 50 supercomputers in the world. We report important discoveries about near-cliques and triangles on several real-world graphs, including a snapshot of the Twitter social network (*56Gb*, 2 billion edges) and the "YahooWeb" dataset, one of the largest publicly available graphs (*120Gb*, 1.4 billion nodes, 6.6 billion edges).

Index Terms—Spectral Analysis; MapReduce; Hadoop; HEigen; Graph Mining

1 INTRODUCTION

Graphs with billions of nodes and edges, or *billion-scale* graphs, are becoming common; Facebook boasts about 0.8 billion active users, who-calls-whom networks can reach similar sizes in large countries, and web crawls can easily reach billions of nodes. Given a billion-scale graph, how can we find near-cliques (a set of tightly connected nodes), the count of triangles, and related graph properties? As we discuss later, triangle counting and related expensive operations can be computed quickly, provided we have the first several eigenvalues and eigenvectors. In general, spectral analysis is a fundamental tool not only for graph mining, but also for other areas of data mining. Eigenvalues and eigenvectors are at the heart of numerous algorithms such as triangle counting [1], singular value decomposition (SVD) [2], [3], spectral clustering [4], [5], [6], Principal Component Analysis (PCA) [7], Multi Dimensional Scaling (MDS) [8], [9], Latent Semantic Indexing (LSI) [10], and tensor analysis [11], [12], [13], [14]. Despite their importance, existing eigensolvers do not scale well. As described in Section 7, the maximum order and size of input matrices feasible for these solvers are million-scales.

In this paper, we discover patterns on near-cliques and triangles, on several real-world graphs including a Twitter dataset (*56Gb*, over 2 billion edges) and the "YahooWeb" dataset, one of the largest publicly available graphs (*120Gb*, 1.4 billion nodes, 6.6 billion edges). To enable discoveries, we propose HEIGEN, an eigensolver for billion-scale, sparse symmetric matrices built on the top of HADOOP, an open-source MAPREDUCE framework. Our contributions are the following:

1) **Effectiveness:** With HEIGEN we analyze billion-scale real-world graphs and report discoveries, including a high triangle vs. degree ratio for adult sites and web pages that participate in billions of triangles.
2) **Careful Design:** We choose among several serial algorithms and selectively parallelize operations for better efficiency.
3) **Scalability:** We use the HADOOP platform for its excellent scalability and implement several optimizations for HEIGEN, such as cache-based multiplications and skewness exploitation. This results in linear scalability in the number of edges, the same accuracy as standard eigensolvers for small matrices, and more than a 76× performance improvement over a naive implementation.

Due to our focus on scalability, HEIGEN can handle sparse symmetric matrices which correspond to graphs with *billions of nodes and edges*, surpassing the capability of previous eigensolvers (e.g. [15] [16]) by more than *1,000*×. Note that HEIGEN is different from Google's PageRank algorithm [17] since HEIGEN computes the top k eigenvectors while PageRank computes only the *first* eigenvector. Designing top k eigensolver is much more difficult and subtle than designing the first eigensolver, as we will see in Section 4. With this powerful tool we are able to study several billion-scale graphs, and we report fascinating

• *U Kang, Brendan Meeder, Evangelos E. Papalexakis, and Christos Faloutsos are with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 15213, USA.*
E-mail: {ukang,bmeeder,epapalex,christos}@cs.cmu.edu

TABLE 1
Order and size of networks.

Name	Nodes	Edges	Description
YahooWeb	1,413 M	6,636 M	WWW pages in 2002
Twitter	62.5 M	2,780 M	who follows whom in 2009/11
LinkedIn	7.5 M	58 M	person-person in 2006
Wikipedia	3.5 M	42 M	doc-doc in 2007/02
Kronecker	177 K	1,977 M	synthetic graph
	120 K	1,145 M	
	59 K	282 M	
WWW- Barabasi	325 K	1,497 K	Web pages inside nd.edu
Epinions	75 K	508 K	who trusts whom

patterns on the near-cliques and triangle distributions in Section 2.

The HEIGEN algorithm (implemented in HADOOP) is available at
`http://www.cs.cmu.edu/~ukang/HEIGEN`. The rest of the paper is organized as follows. In Section 2 we presents the discoveries in real-world, large scale graphs. Section 3 explains the design decisions that we considered for selecting the best sequential method. Section 4 describes HEIGEN, our proposed eigensolver. Section 5 explains additional uses of HEIGEN for interesting eigenvalue based algorithms. Section 6 shows the performance results of HEIGEN. After describing previous works in Section 7, we conclude in Section 8.

2 DISCOVERIES

In this section, we show discoveries on billion-scale graphs using HEIGEN. The discoveries include spotting near-cliques, finding triangles, and eigen power-laws. The graphs we used in this and Section 6 are described in Table 1. [1] In all the experiments for this section, we used top 10 eigenvalues and eigenvectors computed from 50 iterations of HEIGEN.

2.1 Spotting Near-Cliques

In a large, sparse network, how can we find tightly connected nodes, such as those in near-cliques or bipartite cores? Surprisingly, eigenvectors can be used for this purpose [18]. Given an adjacency matrix A and its SVD $A = U\Sigma V^T$, an *EE-plot* is defined to be the scatter plot of the vectors U_i and U_j for any i and j. EE-plots of some real-world graphs contain clear separate lines (or 'spokes'), and the nodes with the largest values in each spoke distinguish themselves from the other nodes by forming near-cliques or bipartite cores. Figures 1 shows several EE-plots and spyplots (i.e., adjacency matrix of induced subgraph) of the top 100 nodes in top eigenvectors of YahooWeb graph.

1. Twitter: http://www.twitter.com/
Wikipedia: http://www.cise.ufl.edu/research/sparse/matrices/
Kronecker: http://www.cs.cmu.edu/~ukang/dataset
Other graphs are either under NDA or not public.

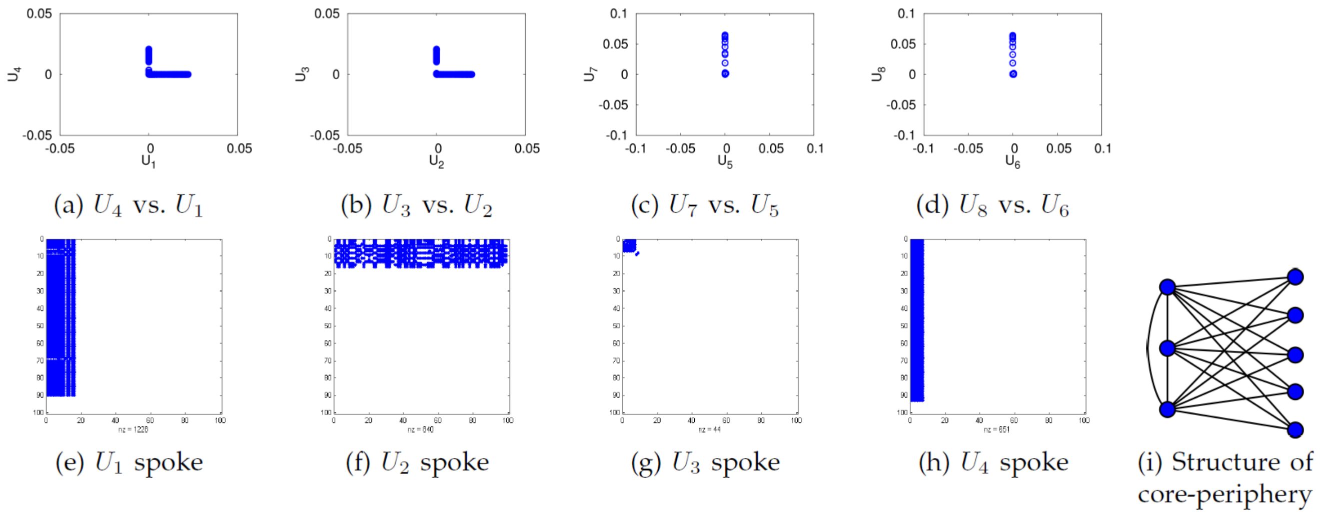
In Figure 1 (a) - (d), we observe clear 'spokes,' or outstanding nodes, in the top eigenvectors. Moreover, in Figure 1 (e), (f), and (h), the top 100 nodes with largest values in U_1, U_2, and U_4, respectively, form a 'core-periphery' (complete bipartite graph with nodes in one side forming a clique) as depicted in Figure 1 (i). Another observation is that the top seven nodes shown in Figure 1 (g) belong to `indymedia.org` which is the site with the maximum number of triangles as shown in Figure 3. We also note that the nodes in (e) - (h) highly overlap: the number of distinct nodes is 109.

In the WWW-Barabasi graph of Figure 2, we also observe spokes in the top eigenvectors. The spokes from the top four eigenvectors form near-cliques, and the union of them (329 nodes) clearly identify three tightly connected communities in Figure 2 (i).

Observation 1 (Eigenspokes): EE-plots of real graphs show clear spokes. Additionally, the extreme nodes in the spokes belong to cliques or core-peripheries. □

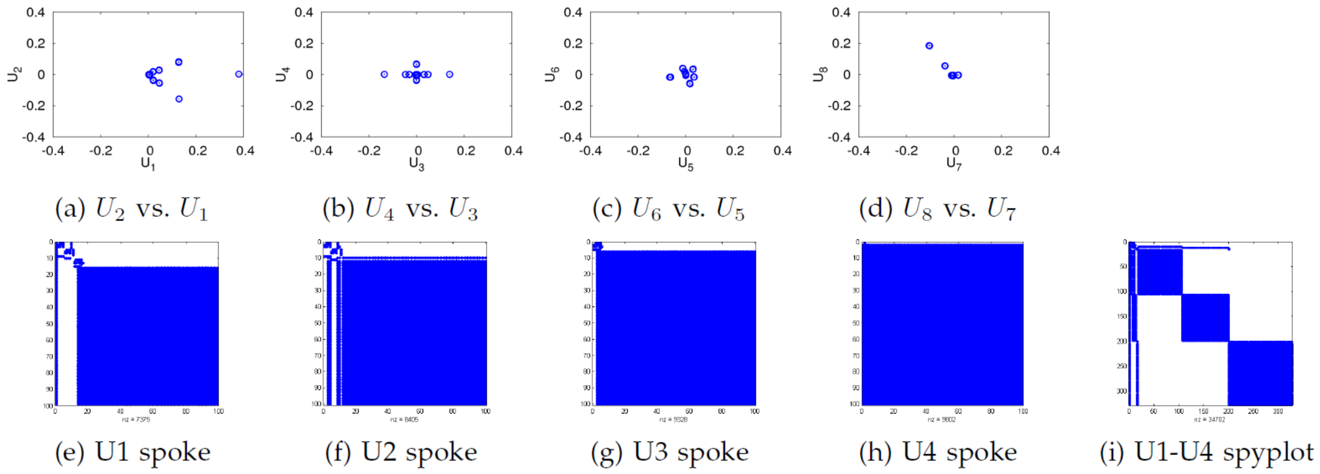
2.2 Triangle Counting

Given a particular node in a graph, how are its neighbors connected? Do they form stars? Cliques? The above questions about the community structure of networks can be answered by studying triangles (three nodes which are connected to each other). However, directly counting triangles in graphs with billions of nodes and edges is prohibitively expensive [19]. Fortunately, we can approximate triangle counts with high accuracy using HEIGEN by exploiting the connection of triangle counting to eigenvalues [20]. In a nutshell, the total number of triangles in a graph is related to the sum of cubes of eigenvalues, and the first few eigenvalues provide extremely good approximations. A slightly more elaborate analysis approximates the number of triangles in which a node participates, using the cubes of the first few eigenvalues and the corresponding eigenvectors. Specifically, the total number of triangles $\Delta(G)$ of a graph G is $\Delta(G) = \frac{1}{6}\sum_{i=1}^{n}\lambda_i^3$, and the number of triangles Δ_i that a node i is participating in is $\Delta_i = \frac{1}{2}\sum_{j=1}^{n}\lambda_j^3 u_j[i]^2$ where λ_j is the jth eigenvalue and $u_j[i]$ is the ith element of the jth eigenvector of the adjacency matrix


(a) U_4 vs. U_1 (b) U_3 vs. U_2 (c) U_7 vs. U_5 (d) U_8 vs. U_6

(e) U_1 spoke (f) U_2 spoke (g) U_3 spoke (h) U_4 spoke (i) Structure of core-periphery

Fig. 1. EE-plots and spyplots from YahooWeb. **(a)-(d)**: EE-plots showing the scores of nodes in the ith eigenvector U_i vs. in the jth eigenvector U_j. Notice the clear 'spokes' in top eigenvectors signify the existence of a strongly related group of nodes in near-cliques or core-periphery (complete bipartite graph with nodes in one side forming a clique) as depicted in (i). **(e)-(h)**: spyplots (adjacency matrices of induced subgraphs) of the top 100 largest scoring nodes from each eigenvector. Notice that we see a near clique in U_3, and core-peripheries in U_1, U_2, and U_4. **(i)**: the structure of 'core-periphery' in (e), (f), and (h).


(a) U_2 vs. U_1 (b) U_4 vs. U_3 (c) U_6 vs. U_5 (d) U_8 vs. U_7

(e) U1 spoke (f) U2 spoke (g) U3 spoke (h) U4 spoke (i) U1-U4 spyplot

Fig. 2. EE-plots and spyplots of WWW-Barabasi. **(a)-(d)**: EE-plots showing the scores in the ith eigenvector U_i vs. in the jth eigenvector U_j. Notice the 'spokes' in top eigenvectors which signify the cliques shown in the second row. **(e)-(h)**: Spyplots (adjacency matrices of induced subgraphs) from the top 100 largest scoring nodes from each eigenvector. Notice the cliques in all the plots. **(i)**: Spyplots of the union of nodes in the top 4 spokes. Notice the 3 cliques of sizes 90, 100, and 130.

of G. The top k eigenvalues can give highly accurate approximations to the number of triangles since the top eigenvalues dominate the cubic sum given the power-law relation of eigenvalues [21], which we also observe in Section 2.3.

Using the top k eigenvalues computed with HEIGEN, we analyze the distribution of triangle counts of real graphs including LinkedIn, Twitter, and YahooWeb graphs in Figure 3. We first observe that there exist several nodes with extremely large triangle counts. In Figure 3 (b), Barack Obama is the person with the fifth largest number of participating triangles, and has many more than other U.S. politicians. In Figure 3 (c), the web page `lists.indymedia.org` contains the largest number of triangles; this page is a list of mailing lists which apparently point to each other.

We also observe regularities in triangle distributions and note that the beginning part of the distributions follows a power-law.

Observation 2 (Triangle power law): The beginning part of the triangle count distribution of real graphs

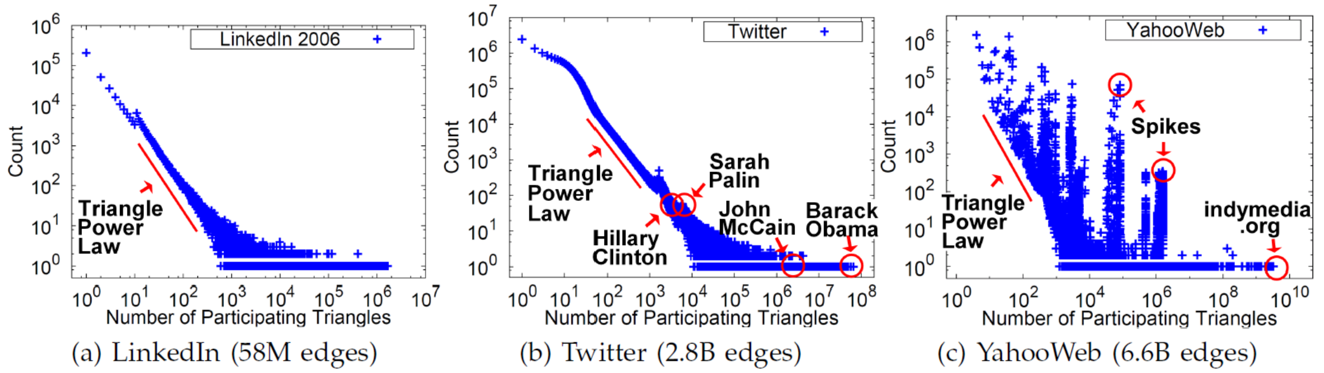


Fig. 3. The distribution of the number of participating triangles of real graphs. In general, they obey the "triangle power-law." In the Twitter plot, some well-known U.S. politicians are circled; among them Barack Obama has the largest number of triangles. In the YahooWeb graph, we observe several anomalous spikes which possibly come from cliques.

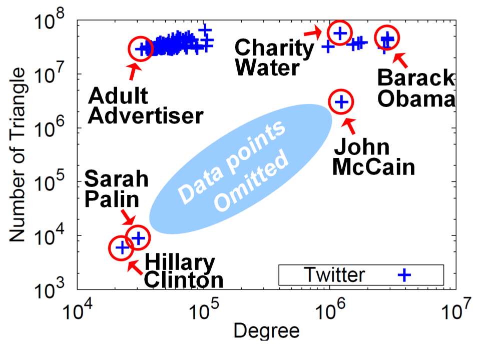


Fig. 4. The number of participating triangles vs. degree of some 'celebrities' (rest: omitted, for clarity) in Twitter accounts. Also shown are accounts of adult sites which have smaller degree, but belong to an abnormally large number of triangles (= many, well connected followers - probably, 'robots').

follows a power-law. □

In the YahooWeb graph in Figure 3 (c), we observe many spikes. One possible explanation for these spikes is that they come from cliques: a k-clique generates k nodes with $\binom{k-1}{2}$ triangles.

Observation 3 (Spikes in triangle distribution): In the Web graph, there exist several spikes which possibly come from cliques. □

The rightmost spike in Figure 3 (c) contains 125 web pages each of which has about 1 million triangles in their neighborhoods. They all belong to the news site `ucimc.org`, and are connected to a tightly coupled group of pages.

Triangle counts exhibit even more interesting patterns when combined with the degree information as shown in the triangle-degree plot of Figure 4. In general, the triangle and the degree are linearly correlated in the log-log plot [20]; for example the celebrities in Figure 4 have similar mild ratios for the triangles and the degrees. However, accounts for some adult sites have extremely well connected followers which make the ratio very high. Degree-triangle plots can be used to spot and eliminate harmful accounts such as those of adult advertisers and spammers. We note that not all of the high triangle vs. degree ratio nodes are suspicious; however, they should be given high priority for possible investigation.

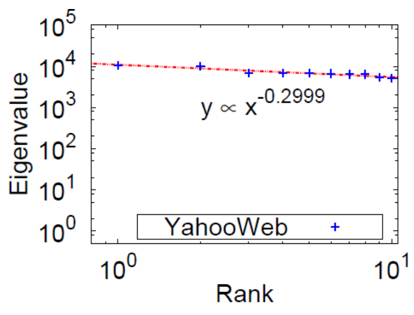
Observation 4 (Anomalous Triangles vs. Degree Ratio): In Twitter, accounts from some adult advertisers have very high triangles vs. degree ratio compared to other regular accounts. □

2.3 Eigen Exponent

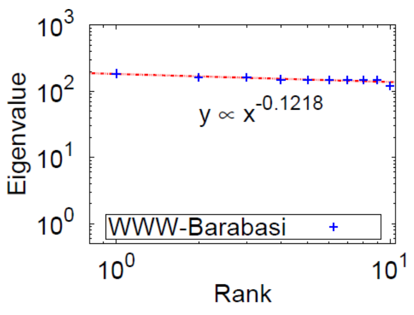
The power-law relationship $\lambda_i \propto i^{\varepsilon}$ of eigenvalues λ_i vs. rank i has been observed in Internet topology graphs with up to 4,389 nodes [21]. Will the same power-law be observed in up to 300,000× larger graphs? The scree plots in Figure 5 show the answer. Note that all plots have correlation coefficients equal to -0.94 or better, except for the WWW-Barabasi with the correlation coefficient -0.84.

Observation 5 (Power-law scree plots): For all real graphs of Figure 5, the scree plots indicate power laws. Most of the graphs have the correlation coefficients -0.94 or better. □

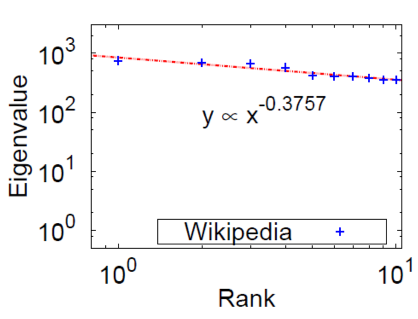
The difference of the slopes between YahooWeb and WWW-Barabasi graphs means that the larger Web



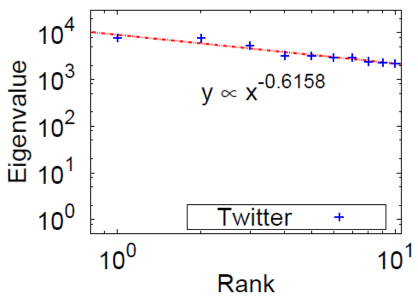
(a) YahooWeb
Corr. Coefficient: -0.943



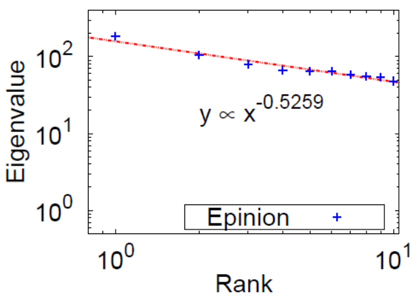
(b) WWW-Barabasi
Corr. Coefficient: -0.842



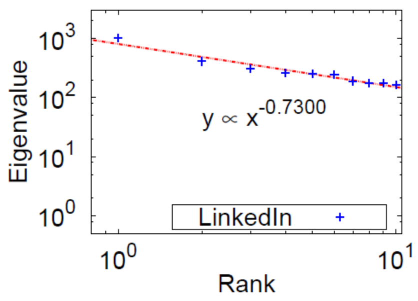
(c) Wikipedia
Corr. Coefficient: -0.946



(d) Twitter
Corr. Coefficient: -0.961



(e) Epinions
Corr. Coefficient: -0.974



(f) LinkedIn
Corr. Coefficient: -0.972

Fig. 5. The scree plot: absolute eigenvalue vs. rank in log-log scale. Notice that the similar power-laws are observed in various real graphs.

graph (YahooWeb) has a more skewed eigenvalue distribution, with the small set of eigenvalues dominating most of the spectra compared to the smaller Web graph (WWW-Barabasi).

All of the above observations need a fast, scalable eigensolver. This is exactly what HEIGEN does, and we describe our proposed design next.

3 BACKGROUND - SEQUENTIAL ALGORITHMS

Our goal is an eigensolver that finds the top k eigenvalues of a billion-scale matrix. Our natural choice of parallel platform is HADOOP, since it has been successfully used for processing Web-scale data and many graph mining tasks also (see [22], [23], [24], [25], [26], [27]). We limit our attention to symmetric matrices due to the computational difficulty since even the best method for non-symmetric eigensolver requires significantly heavier computations than the symmetric case [28].

The problem of finding the eigenvalues of a matrix, however, is inherently difficult since it essentially boils down to finding the roots of a high-degree polynomial which may not have the general solution. Designing the parallel eigensolver algorithm is even more complicated since it requires a careful choice of operations that could be performed well in parallel. In this section, we review some of the major sequential eigensolver algorithms and show the important design decisions that guided our choice of the best sequential method for parallel eigensolvers for very large graphs. Table 2 lists the symbols used in this paper. For indexing elements of a matrix, we use $A[i,j]$ for (i,j)th element of A, $A[i,:]$ for ith row of A, and $A[:,j]$ for jth column of A.

3.1 Power Method

The simplest and most popular way of finding the *first* eigenvector of a matrix is the Power method. The *first* eigenvector is the one corresponding to the largest eigenvalue of the matrix. In the Power method, the input matrix A is multiplied with the initial random vector b multiple times to compute the sequence of vectors $Ab, A(Ab), A(A^2b), \ldots$ which converges to the first eigenvector of A.

The Power method is attractive since it requires only matrix-vector multiplications, which are carried out efficiently in many parallel platforms including HADOOP [27]. Furthermore, it is one of the ways of computing the PageRank of a graph [17]. However, the main drawback of the Power method in the present context is that it is very restrictive, since it computes only the first eigenvector. Other variants of the power method, such as *shifted inverse iteration* and *Rayleigh quotient iteration* also have the same limitation. Therefore, we need to find a better method which can find top k eigenvalues and eigenvectors.

TABLE 2
Table of symbols.

Symbol	Definition
n	order of input matrix
m	number of iterations in Lanczos
A, M	n-by-n input matrix
A_s, M_s	n-by-m input matrix, $n \gg m$
y, x	n-vector
x_s	m-vector, $n \gg m$
α, β	a real number
$\|\|y\|\|$	L2 norm of the vector y
$\|\|T\|\|$	induced matrix L2 norm of the matrix T which is the largest singular value of T
e_m	a vector whose mth element is 1, while other elements are 0
$EIG(A)$	outputs QDQ^T by symmetric eigen decomposition
ϵ	machine epsilon: upper bound on the relative computation error

Shortcomings: Power method computes only the first eigenvector.

3.2 Simultaneous Iteration (or QR algorithm)

The simultaneous iteration (or QR algorithm, which is essentially the same) is an extension of Power method in the sense that it applies the Power method to several vectors at once. It can be shown that the orthogonal bases of the vectors converge to top k eigenvectors of the matrix [28]. The main problem of the simultaneous iteration is that it requires several large matrix-matrix multiplications which are prohibitively expensive for billion-scale graphs. Therefore, we restrict our attention to algorithms that require only matrix-vector multiplications.

Shortcomings: Simultaneous iteration is too expensive for billion-scale graphs due to large matrix-matrix multiplications.

3.3 Lanczos-NO: No Orthogonalization

The next method we consider is the basic Lanczos algorithm [29] which we henceforth call Lanczos-NO (No Orthogonalization). Lanczos-NO method is attractive since it can find the top k eigenvalues of sparse, symmetric matrix, with its most costly operation being the matrix-vector multiplication.

Overview - Intuition. The Lanczos-NO algorithm is a clever improvement over the Power method. Like the Power method,

- it requires several (m) matrix-vector multiplications, that can easily be done with HADOOP
- then it generates a dense, but skinny matrix ($n \times m$, with $n \gg m$)
- it computes a small, sparse square $m \times m$ matrix, whose eigenvalues are good approximations to the required eigenvalues
- and then computes the top k eigenvectors ($k < m$), also with HADOOP-friendly operations.

Thus, all the expensive steps can be easily done with HADOOP. Next we provide more details on Lanczos-NO, which can be skipped on first glance.

Details. The Lanczos-NO algorithm is a clever extension of the Power method. In the Power method, the intermediate vectors $A^k b$ are discarded for the final eigenvector computation. In Lanczos-NO, the intermediate vectors are used for constructing orthonormal bases of the so-called *Krylov subspace* K_m, which is defined as

$$K_m = < b, Ab, ..., A^{m-1}b > .$$

The orthonormal bases are constructed by creating a new vector which is orthogonal to all previous bases, as in Gram-Schmidt orthogonalization. Therefore, Lanczos-NO can be summarized as an iterative algorithm which constructs orthonormal bases for successive Krylov subspaces. Specifically, Lanczos-NO with m iterations computes the Lanczos-NO factorization which is defined as follows:

$$AV_m = V_m T_m + f_m e_m^T,$$

where $A^{n\times n}$ is the input matrix, $V_m^{n\times m}$ contains the m orthonormal bases as its columns, $T_m^{m\times m}$ is a tridiagonal matrix that contains the coefficients for the orthogonalization, f_m is a new n-vector orthogonal to all columns of V_m, and e_m is a vector whose mth element is 1, while other elements are 0. Here, m (the number of matrix-vector multiplication) is much smaller than n (the order of the input matrix): e.g., for billion-scale graphs, $n = 10^9$, and $m = 20$. The Lanczos-NO iteration is shown in Algorithm 1.

After m iterations, the V_m matrix and T_m matrices are constructed (T_m is built by $T_m[i,i] \leftarrow \alpha_i$, and $T_m[i,i+1] = T_m[i+1,i] \leftarrow \beta_i$). The eigenvalues of T_m are called the Ritz values, and the columns of $V_m Y$, where Y contains the eigenvector of T_m in its columns, are called the Ritz vectors which are constructed by Algorithm 2. The Ritz values and the Ritz vectors are good approximations of the eigenvalues and the eigenvectors of A, respectively [30]. The computation of the eigenvalues of T_m can be done quickly with direct algorithms such as QR since

Algorithm 1: Lanczos-NO (No Orthogonalization)

```
Input: Matrix $A^{n \times n}$,
    random $n$-vector $b$,
    number of steps $m$
Output: Orthogonal matrix $V_m^{n \times m} = [v_1 ... v_m]$,
    coefficients $\alpha[1..m]$ and $\beta[1..m-1]$
1: $\beta_0 \leftarrow 0$, $v_0 \leftarrow 0$, $v_1 \leftarrow b/||b||$;
2: for $i = 1, ..m$ do
3:   $v \leftarrow Av_i$;
4:   $\alpha_i \leftarrow v_i^T v$;
5:   $v \leftarrow v - \beta_{i-1} v_{i-1} - \alpha_i v_i$; // make a new basis
6:   $\beta_i \leftarrow ||v||$;
7:   if $\beta_i = 0$ then
8:     break for loop;
9:   end if
10:  $v_{i+1} \leftarrow v/\beta_i$;
11: end for
```

Algorithm 2: Compute Top k Ritz Vectors

```
Input: Orthogonal matrix $V_m^{n \times m}$,
    coefficients $\alpha[1..m]$ and $\beta[1..m-1]$
Output: Ritz-vector $R_k^{n \times k}$
1: $T_m \leftarrow$ (build a tri-diagonal matrix from $\alpha$ and $\beta$);
2: $QDQ^T \leftarrow EIG(T_m)$;
3: $\lambda_{1..k} \leftarrow$ (top $k$ eigenvalues from $D$);
4: $Q_k \leftarrow$ ($k$ columns of $Q$ corresponding to $\lambda_{1..k}$);
5: $R_k \leftarrow V_m Q_k$;
```

the matrix is very small (e.g., 20 by 20). For details, see [30].

The problem of Lanczos-NO is that some eigenvalues jump up to the next eigenvalues, thereby creating spurious eigenvalues. We'll see the solution to this problem in the next section.

Shortcomings: Lanczos-NO outputs spurious eigenvalues.

4 Proposed Method

In this section we describe HEigen, a parallel algorithm for computing the top k eigenvalues and eigenvectors of symmetric matrices in MapReduce.

4.1 Summary of the Contributions

Efficient top k eigensolvers for billion-scale graphs require careful algorithmic considerations. The main challenge is to carefully design algorithms that work well on distributed systems and exploit the inherent structure of data, including block structure and skewness, in order to be efficient. We summarize the algorithmic contributions here and describe each in detail in later sections.

1) **Careful Algorithm Choice:** We carefully choose a sequential eigensolver algorithm that is efficient for MapReduce and gives accurate results.
2) **Selective Parallelization:** We group operations into expensive and inexpensive ones based on input sizes. Expensive operations are done in parallel for scalability, while inexpensive operations are performed on a single machine to avoid extra overhead of parallel execution.
3) **Blocking:** We reduce the running time by decreasing the input data size and the amount of network traffic among machines.
4) **Exploiting Skewness:** We decrease the running time by exploiting the skewness of data.

4.2 Careful Algorithm Choice

In Section 3, we considered three algorithms that are not tractable for analyzing billion-scale graphs with MapReduce. Fortunately, there is an algorithm suitable for such a purpose. Lanczos-SO (Selective Orthogonalization) improves on the Lanczos-NO by selectively reorthogonalizing vectors instead of performing full reorthogonalizations.

The main idea of Lanczos-SO is as follows: we start with a random initial basis vector b which comprises a rank-1 subspace. For each iteration, a new basis vector is computed by multiplying the input matrix with the previous basis vector. The new basis vector is then orthogonalized against the last two basis vectors and is added to the previous rank-$(m-1)$ subspace, forming a rank-m subspace. Let m be the number of the current iteration, Q_m be the $n \times m$ matrix whose ith column is the ith basis vector, and A be the matrix whose eigenvalues we seek to compute. We also define $T_m = Q_m^T A Q_m$ to be a $m \times m$ matrix. Then, the eigenvalues of T_m are good approximations of the eigenvalues of A . Furthermore, multiplying Q_m by the eigenvectors of T_m gives good approximation of the eigenvectors of A. We refer to [28] for further details.

If we used the exact arithmetic, the newly computed basis vector would be orthogonal to all previous basis vectors. However, rounding errors from floating-point calculations compound and result in the loss of orthogonality. This is the cause of the spurious eigenvalues in Lanczos-NO. Orthogonality can be recovered once the new basis vector is fully re-orthogonalized to all previous vectors. However, this operation is quite expensive as it requires $O(m^2)$ re-orthogonalizations, where m is the number of iterations. A faster approach uses a quick test (line 10 of Algorithm 3) to selectively choose vectors that need to be re-orthogonalized to the new basis [31]. This selective-reorthogonalization idea is shown in Algorithm 3.

The Lanczos-SO has all the properties that we need: it finds the top k largest eigenvalues and eigenvectors, it produces no spurious eigenvalues, and its most expensive operation, a matrix-vector multiplication, is tractable in MapReduce. Therefore, we pick Lanczos-

Algorithm 3: Lanczos-SO (Selective Orthogonalization)

```
Input: Matrix $A^{n \times n}$,
    random $n$-vector $b$,
    maximum number of steps $m$,
    error threshold $\epsilon$,
    number of eigenvalues $k$
Output: Top $k$ eigenvalues $\lambda_{1..k}$,
    eigenvectors $U^{n \times k}$
1: $\beta_0 \leftarrow 0$, $v_0 \leftarrow 0$, $v_1 \leftarrow b/||b||$;
2: for $i = 1..m$ do
3:   $v \leftarrow Av_i$; // Find a new basis vector
4:   $\alpha_i \leftarrow v_i^T v$;
5:   $v \leftarrow v - \beta_{i-1}v_{i-1} - \alpha_i v_i$; // Orthogonalize
     against two previous basis vectors
6:   $\beta_i \leftarrow ||v||$;
7:   $T_i \leftarrow$ (build tri-diagonal matrix from $\alpha$ and $\beta$);
8:   $QDQ^T \leftarrow EIG(T_i)$; // Eigen decomposition of
     $T_i$
9:   for $j = 1..i$ do
10:    if $\beta_i|Q[i,j]| \leq \sqrt{\epsilon}||T_i||$ then
11:      $r \leftarrow V_i Q[:,j]$;
12:      $v \leftarrow v - (r^T v)r$; // Selectively
         orthogonalize
13:    end if
14:  end for
15:  if ($v$ was selectively orthogonalized) then
16:    $\beta_i \leftarrow ||v||$; // Recompute normalization
       constant $\beta_i$
17:  end if
18:  if $\beta_i = 0$ then
19:    break for loop;
20:  end if
21:  $v_{i+1} \leftarrow v/\beta_i$;
22: end for
23: $T \leftarrow$ (build tri-diagonal matrix from $\alpha$ and $\beta$);
24: $QDQ^T \leftarrow EIG(T)$; // Eigen decomposition of $T$
25: $\lambda_{1..k} \leftarrow$ top k diagonal elements of D; //
    Compute eigenvalues
26: $U \leftarrow V_m Q_k$; // Compute eigenvectors. $Q_k$ is the
    set of columns of $Q$ corresponding to $\lambda_{1..k}$
```

SO as our choice of the sequential algorithm for parallelization.

4.3 Selective Parallelization

Among many sub-operations in Algorithm 3, which operations should we parallelize? A naive approach is to parallelize all the operations; however, some operations run more quickly on a single machine rather than on multiple machines in parallel. The reason is that the overhead incurred by using MAPREDUCE exceeds gains made by parallelizing the task; simple tasks where the input data is very small are carried out faster on a single machine. Thus, we divide the sub-operations into two groups: those to be parallelized and those to be run in a single machine. Table 3 summarizes our choice for each sub-operation. Note that the last two operations in the table can be done with a single-machine standard eigensolver since the input matrices are tiny; they have m rows and columns, where m is the number of iterations.

4.4 Blocking

Minimizing the volume of information exchanged between nodes is important to designing efficient distributed algorithms. In HEIGEN, we decrease the amount of network traffic by using the block-based operations. Normally, one would put each edge "(source, destination)" in one line; HADOOP treats each line as a data element for its mapper functions. Instead, we divide the adjacency matrix into square blocks (and, of course, the corresponding vectors also into blocks), and put the edges of each block on a single line [27], [32]. This makes the mapper functions a bit more complicated to process blocks, but it saves significant transfer time of data over the network. We use these edge-blocks and the vector-blocks for many parallel operations in Table 3, including matrix-vector multiplication, vector update, vector dot product, vector scale, and vector L2 norm. Performing operations on blocks is faster than doing so on individual elements since both the input size and the key space decrease. This reduces the network traffic and sorting time in the MAPREDUCE Shuffle stage. As we will see in Section 6, the blocking decreases the running time by more than 4×.

4.5 Exploiting Skewness: Matrix-Vector Multiplication

HEIGEN uses an adaptive method for sub-operations based on the size of the data. In this section, we describe how HEIGEN implements different matrix-vector multiplication algorithms by exploiting the skewness pattern of the data. There are two matrix-vector multiplication operations in Algorithm 3: the one with a large vector (line 3) and the other with a small vector (line 11).

The first matrix-vector operation multiplies a matrix with a large and dense vector, and thus it requires a two-stage standard MAPREDUCE algorithm by Kang et al. [27]. In the first stage, matrix elements and vector elements are joined and multiplied to produce partial results which are added together to get the result vector in the second stage.

The other matrix-vector operation, however, multiplies with a small vector. HEIGEN uses the fact that the small vector can fit in a machine's main memory, and distributes the small vector to all the mappers using the distributed cache functionality of HADOOP. The advantage of the small vector being available in mappers is that joining edge elements and vector elements can be done inside the mapper,

TABLE 3
Parallelization Choices. The last column (**P?**) indicates whether the operation is parallelized in HEIGEN. Some operations are better to be run in parallel since the input size is very large, while others are better in a single machine since the input size is small and the overhead of parallel execution overshadows its decreased running time.

Operation	Description	Input	P?
$y \leftarrow y + ax$	vector update	Large	Yes
$\gamma \leftarrow x^T x$	vector dot product	Large	Yes
$y \leftarrow \alpha y$	vector scale	Large	Yes
$\|\|y\|\|$	vector L2 norm	Large	Yes
$y \leftarrow M^{n \times n} x$	large matrix-large, dense vector multiplication	Large	Yes
$y \leftarrow M_s^{n \times m} x_s$	large matrix-small vector multiplication ($n \gg m$)	Large	Yes
$A_s \leftarrow M_s^{n \times m} N_s^{m \times k}$	large matrix-small matrix multiplication ($n \gg m > k$)	Large	Yes
$\|\|T\|\|$	matrix L2 norm which is the largest singular value of the matrix	Tiny	No
$EIG(T)$	symmetric eigen decomposition to output QDQ^T	Tiny	No

Algorithm 4: CBMV (Cache-Based Matrix-Vector Multiplication) for HEIGEN

```
Input: Matrix $M = \{(id_{src}, (id_{dst}, mval))\}$,
    vector $x = \{(id, vval)\}$
Output: Result vector $y$
 1: Map(key $k$, value $v$, Vector $x$): // Multiply matrix elements and the vector $x$
 2: $id_{src} \leftarrow k$;
 3: $(id_{dst}, mval) \leftarrow v$;
 4: Output($id_{src}, (mval \times x[id_{dst}])$); // Multiply and output partial results
 5:
 6: Reduce(key $k$, values $V$[]): // Sum up partial results
 7: $sum \leftarrow 0$;
 8: for $v \in V$ do
 9:     $sum \leftarrow sum + v$;
10: end for
11: Output($k, sum$); // Output a vector element
```

and thus the first stage of the standard two-stage matrix-vector multiplication can be omitted. In this one-stage algorithm the mapper joins matrix elements and vector elements to make partial results, and the reducer adds up the partial results. The pseudo code of this algorithm, which we call CBMV (Cache-Based Matrix-Vector multiplication), is shown in Algorithm 4. We want to emphasize that this operation cannot be performed when the vector is large, as is the case in the first matrix-vector multiplication (line 3 of Algorithm 3). The CBMV is faster than the standard method by 57× as described in Section 6.

4.6 Exploiting Skewness: Matrix-Matrix Multiplication

Skewness can also be exploited to efficiently perform matrix-matrix multiplication (line 26 of Algorithm 3). In general, matrix-matrix multiplication is very expensive. A standard, yet naive, way of multiplying two matrices A and B in MAPREDUCE is to multiply $A[:, i]$ and $B[i, :]$ for each column i of A and sum the resulting matrices. This algorithm, which we call direct Matrix-Matrix multiplication (MM), is very inefficient since it generates huge matrices which are summed up many times. Fortunately, when one of the matrices is very small, we may exploit the skewness to come up with an efficient MAPREDUCE algorithm. This is exactly the case in HEIGEN; the first matrix is very large, and the second is very small. The main idea is to distribute the second matrix using the distributed cache functionality in HADOOP, and multiply each element of the first matrix with the corresponding rows of the second matrix. We call the resulting algorithm Cache-Based Matrix-Matrix multiplication, or CBMM. There are other alternatives to matrix-matrix multiplication: one can decompose the second matrix into column vectors and iteratively multiply the first matrix with each of these vectors. We call the algorithms, introduced in Section 4.5, Iterative Matrix-Vector multiplications (IMV) and Cache-Based iterative Matrix-Vector multiplications (CBMV). The difference between CBMV and IMV is that CBMV uses cache-based operations while IMV does not. As we will see in Section 6, the best method, CBMM, is faster than naive methods by 76×.

4.7 Analysis

We analyze the time and the space complexities of HEIGEN. In the lemmas below, m is the number of iterations, $|V|$ is the dimension of the matrix, $|E|$ is the number of nonzeros in the matrix, and M is the number of machines.

Lemma 1 (Time Complexity): HEIGEN takes $O(m\frac{|V|+|E|}{M} log\frac{|V|+|E|}{M})$ time.

Proof: The running time of one iteration of HEIGEN is dominated by the matrix-large vector multiplication whose running time is $O(\frac{|V|+|E|}{M} log\frac{|V|+|E|}{M})$. The lemma is proved by multiplying the running time of an iteration by the number of iterations. □

Note that in computing few eigenvalues from a large graph, m can be treated as a constant.

Lemma 2 (Space Complexity): HEIGEN requires $O(|V|+|E|)$ space.

Proof: The maximum storage is required at the intermediate output of the two-stage matrix-vector multiplication where $O(|V|+|E|)$ space is needed. □

5 ADDITIONAL USES OF HEIGEN

The HEIGEN algorithm can be readily extended to other eigenvalue-based algorithms. In this section, we describe large-scale algorithms for Singular Value Decomposition (SVD), HITS, and spectral clustering based on HEIGEN.

5.1 HEIGEN Gives SVD

Given any matrix A, the Singular Value Decomposition (SVD) gives the factorization

$$A = U\Sigma V^T,$$

where U and V are unitary matrices (i.e. square matrices that satisfy $U^TU = UU^T = I$ with I being the identity matrix), and Σ is a diagonal (if A is square) or a rectangular diagonal matrix (if A is rectangular), whose diagonal entries are real and non-negative, and are called *singular values* of A.

The SVD is a very powerful tool in analyzing graphs as well as matrices [2], [3]; some of its applications include optimal matrix approximation in the least squares sense [33], Principal Component Analysis [7], clustering [34] (more specifically a relaxed version of the well known k-means clustering problem) and Information Retrieval/Latent Semantic Indexing [10].

HEIGEN can be extended to SVD of both symmetric and asymmetric matrices.

Symmetric Matrix. For a symmetric matrix A, the singular values of A are the absolute eigenvalues of A, and the singular vectors and the eigenvectors of A are the same up to signs. Thus, given an eigen decomposition $A = U\Lambda U^T$ computed by HEIGEN, we get the SVD

$$\begin{aligned} A &= U\Lambda U^T \\ &= U\Sigma S U^T \\ &= U\Sigma(US)^T, \end{aligned}$$

where $\Lambda = \Sigma S$, Σ is the diagonal matrix whose element $\Sigma(i,i)$ contains the absolute value of $\Lambda(i,i)$, and S is the diagonal matrix whose (i,i)-th element $S(i,i)$ is 1 if $\Lambda(i,i) \geq 0$, and -1 otherwise.

Asymmetric Matrix. For an asymmetric matrix $A^{n\times p}$, the standard method to compute the SVD $A = U\Sigma V^T$ is to build a symmetric $(n+p)\times(n+p)$ matrix

Algorithm 5: Standard SVD on asymmetric matrix using HEIGEN

```
Input: Matrix $A^{n\times p}$,
    number of singular values $k$
Output: Top $k$ singular values $\sigma[1..k]$,
    left singular vectors $U^{n\times k}$,
    right singular vectors $V^{p\times k}$ of $A$
1: $\hat{A} \leftarrow \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$;
2: Apply HEIGEN on $\hat{A}$;
```

$$\hat{A} = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix},$$

and apply HEIGEN on $\hat{A}$ [3]. The SVD (up to signs) of $\hat{A}$ is given by

$$\hat{A} = \begin{bmatrix} \frac{1}{\sqrt{2}}U & -\frac{1}{\sqrt{2}}U \\ \frac{1}{\sqrt{2}}V & \frac{1}{\sqrt{2}}V \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}}U^T & \frac{1}{\sqrt{2}}V^T \\ -\frac{1}{\sqrt{2}}U^T & \frac{1}{\sqrt{2}}V^T \end{bmatrix}.$$

Algorithm 5 shows the algorithm for standard SVD on asymmetric matrix using HEIGEN.

There are two shortcomings in Algorithm 5 which can be improved. First, we need to construct $\hat{A}$ which is $2\times$ larger than the original matrix A. Second, to get k singular values of A, we need to get $2k$ eigenvalues of $\hat{A}$ since there are 2 copies of the same eigenvalues in the eigen decomposition of $\hat{A}$.

Fast SVD for Asymmetric Matrix. We describe a faster SVD method for asymmetric matrices using HEIGEN, with the two main ideas. First, we use the fact that if $A = U\Sigma V^T$ is a SVD of A, then $AA^T = U\Sigma^2U^T$ is a symmetric, positive definite matrix whose eigenvectors are the same as the left singular vectors of A, and eigenvalues are the square of the singular values of A. Thus, HEIGEN on AA^T gives us U and Σ. Having computed U and Σ, we can solve for V^T by $V^T = \Sigma^{-1}U^TA$. Naively applying HEIGEN on AA^T is not desired, however, since AA^T can be much larger and denser than A. Our second idea solves the problem by never materializing the matrix AA^T in applying HEIGEN on AA^T. Note that the input matrix in HEIGEN is used only for the matrix-vector multiplication on line 3 of Algorithm 3. We can efficiently compute the matrix-vector multiplication $(AA^T)v_i$ by $A(A^Tv_i)$, which means to first compute A^Tv_i, and multiply the resulting vector by A, thereby replacing a dense matrix-vector multiplication by two sparse matrix-vector multiplications.

We note that Lanczos-based bidiagonalization method [35], [36], followed by the diagonalization of the bidiagonal matrix, is another viable option for SVD [28].

5.2 HEIGEN Solves Large Scale Systems of Linear Equations

Solving large scale systems of linear equation is a very important task that pertains to almost every scientific discipline. Consider the following system of linear equations:

$$y = Ax$$

where A is of size $m \times p$ and x is the vector of unknowns. In general, this system might have one, infinite or no solution, depending on the dimensions of A (if $m > p$ the system is called *overdetermined*, else if $m < p$ it is called *underdetermined*), and on whether y exists in the column space of A or not. In all the above cases, HEIGEN helps us calculate the only solution (when there exists one), find the best (in terms of the ℓ_2 norm) whenever there are infinite ones, or get the best ℓ_2 approximation of x, when there is no exact solution. For all the aforementioned cases, we call $\hat{x}$ the outcome. It can be shown that the solution $\hat{x}$ is given by $\hat{x} = A^\dagger y$, where $A^\dagger$ is the Moore-Penrose pseudoinverse of A [37] which is defined by $A^\dagger = (A^T A)^{-1} A^T$ for a real matrix A with full column rank. A computationally efficient way to compute the pseudoinverse is to use the SVD. Specifically, given an SVD $A = U \Sigma V^T$ by HEIGEN, the pseudoinverse $A^\dagger$ is given by

$$A^\dagger = V \Sigma^{-1} U^T.$$

Furthermore, $\hat{x} = A^\dagger y = V \Sigma^{-1} U^T y$ can be computed efficiently in HADOOP by three matrix-vector multiplications. Specifically,

$$\hat{x} = Vw,$$

where $w = \Sigma^{-1} z$ and $z = U^T y$. We note that the accuracy of the pseudoinverse computation depends on the rank of the matrix Σ: the higher the rank of Σ, the better the accuracy.

5.3 HEIGEN Gives HITS

HITS [38] is a well-known algorithm to compute the 'hubs' and 'authorities' scores in web pages. Given an adjacency matrix A of web pages, the hub and the authority scores are given by the principal eigenvector of AA^T and $A^T A$, respectively. HEIGEN can give them since the left and the right singular vectors of A are the principal eigenvectors of AA^T and $A^T A$, respectively, as described in Section 5.1.

5.4 HEIGEN Gives Spectral Clustering

Spectral clustering is a popular clustering algorithm on graphs [6]. We consider the two spectral clustering algorithms by Shi et al. [4] and Ng et al. [5], and show how they can be easily computed with HEIGEN. Recall that the main idea of the spectral clustering is to first compute the k smallest eigenvectors of $L_{rw} = D^{-1}L$ (Shi et al. [4]) or $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$ (Ng et al. [5]), respectively, and then run a k-means algorithm. Here, $L = D - A$ is the graph Laplacian matrix where A is a symmetric adjacency matrix of a graph and D is the diagonal matrix computed from A with $D(i,i) = \sum_j A(i,j)$.

Algorithm 6: Spectral Clustering with L_{sym} using HEIGEN

```
Input: Matrix $A^{n \times m}$,
       number of clusters $l$
Output: $l$ clusters $C_{1..l}$
1: Construct $D$;
2: $A' \leftarrow D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$;
3: $[U, \lambda_{1..k}] \leftarrow$ HEIGEN($A'$);
4: $C_{1..l} \leftarrow$ $k$-means on $U$;
```

Algorithm 7: Spectral Clustering with L_{rw} using HEIGEN

```
Input: Matrix $A^{n \times m}$,
       number of clusters $l$
Output: $l$ clusters $C_{1..l}$
1: Construct $D$;
2: $A' \leftarrow D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$;
3: $[U, \lambda_{1..k}] \leftarrow$ HEIGEN($A'$);
4: $U' \leftarrow D^{-\frac{1}{2}} U$;
5: $C_{1..l} \leftarrow$ $k$-means on $U'$;
```

Issues. Applying HEIGEN on the spectral clustering is not straightforward for the following two reasons. First, the spectral clustering algorithms require k *smallest* eigenvectors, while HEIGEN computes k *largest* eigenvectors of a matrix. Second, the L_{rw} is *asymmetric*, while HEIGEN works only on a *symmetric* matrix. However, HEIGEN can be used for these algorithms as we show below. We mildly assume that the input graph for the spectral clustering is connected.

Our solution on L_{sym}. Notice that $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where I is an identity matrix, and $L_{rw} = D^{-1} L = I - D^{-1} A$. It can be shown that $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ and $D^{-1} A$ share the same eigenvalues, and the eigenvalues range from -1 to 1 [6]. Also note that if λ is an eigenvalue of $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, then $1-\lambda$ is an eigenvalue of L_{sym} with the same eigenvector. Thus, the k smallest eigenvalues and eigenvectors of L_{sym} are mapped to the k largest eigenvalues and eigenvectors of $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, which can be computed by HEIGEN. Algorithm 6 shows the spectral clustering with L_{sym} using HEIGEN.

Our solution on L_{rw}. It can be shown that u is an eigenvector of L_{rw} if and only if $D^{\frac{1}{2}} u$ is an eigenvector of L_{sym} with the same eigenvalue [6]. Thus, multiplying $D^{-\frac{1}{2}}$ to the k largest eigenvectors of $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ leads to top k smallest eigenvectors of L_{rw}, as shown in Algorithm 7.

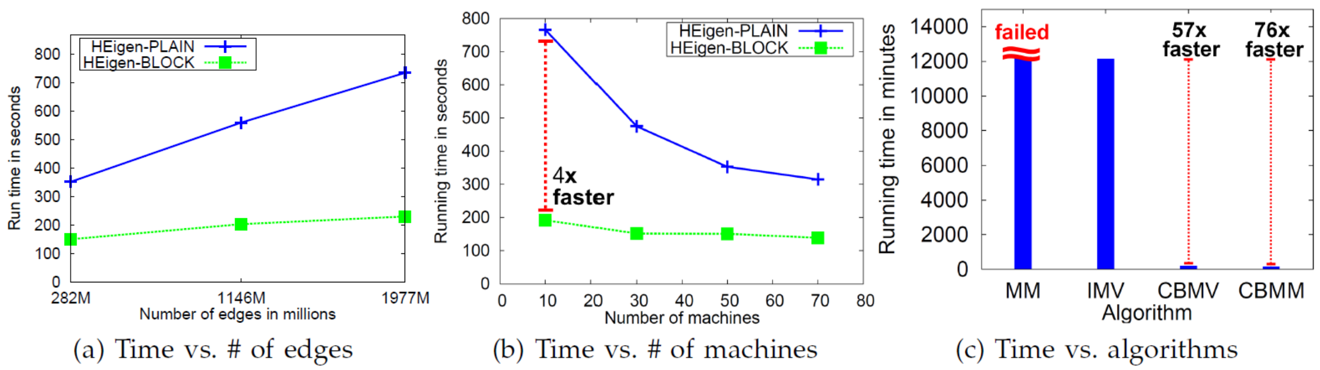


Fig. 6. (**a**) Running time vs. number of edges in 1 iteration of HEIGEN with 50 machines. Notice the near-linear running time proportional to the edges size. (**b**) Running time vs. number of machines in 1 iteration of HEIGEN, on the Kronecker graph with 282M edges. The running time decreases as number of machines increase. (**c**) Comparison of running time between different skewed matrix-matrix and matrix-vector multiplications. For matrix-matrix multiplication, our proposed CBMM outperforms naive methods by at least 76×. The slowest matrix-matrix multiplication algorithm(MM) even didn't finish and the job failed due to excessive data. For matrix-vector multiplication, our proposed CBMV is faster than the naive method by 57×.

6 PERFORMANCE

We present experimental results. We verified the accuracy of HEIGEN on a small graph (Epinions), and the singular values and eigenvalues from HEIGEN and the MATLAB differ at most 0.1%. In addition to this accuracy result, we present answers to the following questions:

- **Scalability:** how well does HEIGEN scale up?
- **Optimizations:** which of our proposed methods give the best performance?

We perform experiments in the Yahoo! M45 HADOOP cluster with total 480 hosts, 1.5 petabytes of storage, and 3.5 terabytes of memory. We use HADOOP 0.20.1. The scalability experiments are performed using synthetic Kronecker graphs [39] since realistic graphs of any size can be easily generated.

6.1 Scalability

Figure 6 (a,b) shows the scalability of HEIGEN-BLOCK, an implementation of HEIGEN that uses blocking, and HEIGEN-PLAIN, an implementation that does not, on Kronecker graphs. For HEIGEN-BLOCK, we used block width 128. Notice that the running time is near-linear in the number of edges and machines. We also note that HEIGEN-BLOCK performs up to 4× faster when compared to HEIGEN-PLAIN. The running time for HEIGEN-BLOCK does not change much after adding more machines; the reason is that after the blocking the graph size is small enough to be processed in 10 machines.

6.2 Optimizations

Figure 6 (c) shows the comparison of running time of the skewed matrix-matrix multiplication and the matrix-vector multiplication algorithms. We used 100 machines for YahooWeb data. For matrix-matrix multiplications, the best method is our proposed CBMM which is 76× faster than repeated naive matrix-vector multiplications (IMV). The slowest matrix-matrix multiplication algorithm did not even finish, and failed due to heavy amounts of intermediate data. For matrix-vector multiplications, our proposed CBMV is faster than the naive method (IMV) by 48×.

7 RELATED WORKS

The related works form two groups, large-scale eigensolvers and MAPREDUCE/HADOOP.

Large-scale Eigensolvers: There are many parallel eigensolvers for large matrices: the work by Zhao et al. [40], HPEC [41], PLANSO [15], ARPACK [42], ScaLAPACK [43], PLAPACK [44] are several examples. All of them are based on MPI with message passing, which has difficulty in dealing with billion-scale graphs. The maximum order of matrices analyzed with these tools is less than 1 million [15], [16], which is far from web-scale data. On the HADOOP side, the Mahout project [45] provides SVD. However, Mahout suffers from two major issues: (a) it assumes that the vector (b, with n=O(billion) entries) fits in the memory of a single machine, and (b) it implements the full re-orthogonalization which is inefficient.

MapReduce and Hadoop: MAPREDUCE is a parallel programming framework for processing web-scale data. MAPREDUCE has two major advantages: (a) it handles data distribution, replication, and load balancing automatically, and furthermore (b) it uses familiar concepts from functional programming. The programmer needs to provide only the *map* and the *reduce* functions. The general framework is as follows [46]: the *map* stage processes input and outputs (key, value) pairs. The *shuffling* stage sorts the map output and distributes them to reducers. Finally, the *reduce* stage processes the values with the same key and outputs the final result. HADOOP [47] is an open source implementation of MAPREDUCE. It also provides a distributed file system (HDFS) and data processing tools such as PIG [48] and Hive . Due to its extreme scalability and ease of use, HADOOP is widely used for large scale data mining [23], [24], [26], [27], [49] . In addition to PIG, there are several high-level language and environments for advanced MAPREDUCE-like systems including SCOPE [50], Sphere [51], and Sawzall [52].

8 CONCLUSION

In this paper we discovered spectral patterns in real-world, billion-scale graphs. This was possible by using HEIGEN, our proposed eigensolver for the spectral analysis of very large-scale graphs. The main contributions are the following:

- **Effectiveness:** We analyze the spectral properties of real world graphs, including Twitter and one of the largest public Web graphs. We report patterns that can be used for anomaly detection and finding tightly-knit communities.
- **Careful Design:** We carefully design HEIGEN to selectively parallelize operations based on how they are most effectively performed.
- **Scalability:** We implement and evaluate a billion-scale eigensolver. Experiments show that HEIGEN scales linearly with the number of edges.

Future research directions include extending the analysis and the algorithms for multi-dimensional matrices, or tensors [12], [13], [53].

ACKNOWLEDGEMENTS

Funding was provided by the U.S. ARO and DARPA under Contract Number W911NF-11-C-0088, by DTRA under contract No. HDTRA1-10-1-0120, and by ARL under Cooperative Agreement Number W911NF-09-2-0053, The views and conclusions are those of the authors and should not be interpreted as representing the official policies, of the U.S. Government, or other funding parties, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. Brendan Meeder is also supported by a NSF Graduate Research Fellowship and funding from the Fine Foundation, Sloan Foundation, and Microsoft.

REFERENCES

[1] C. E. Tsourakakis, "Fast counting of triangles in large real networks without counting: Algorithms and laws," in *ICDM*, 2008, pp. 608–617.

[2] M. Kamel, "Computing the singular value decomposition in image processing," in *Proceedings of Conference on Information Systems*, Princeton, 1984.

[3] M. W. Berry., "Large scale singular value computations," *International Journal of Supercomputer Applications*, 1992.

[4] J. Shi and J. Malik, "Normalized cuts and image segmentation," *CVPR*, 1997.

[5] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *NIPS*, 2002.

[6] U. Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[7] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine Series 6*, vol. 2, no. 11, pp. 559–572, 1901.

[8] J. B. Kruskal and M. Wish, *Multidimensional scaling*. Beverly Hills: SAGE publications, 1978.

[9] B. T. Bartell, G. W. Cottrell, and R. K. Belew, "Latent semantic indexing is an optimal special case of multidimensional scaling," *SIGIR*, pp. 161–167, 1992.

[10] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, Sep. 1990.

[11] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," *KDD*, pp. 374–383, 2006.

[12] T. G. Kolda and J. Sun, "Scalable tensor decompsitions for multi-aspect data mining," *ICDM*, 2008.

[13] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[14] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *TKDD*, vol. 5, no. 2, p. 10, 2011.

[15] K. Wu and H. Simon, "A parallel lanczos method for symmetric generalized eigenvalue problems," *Computing and Visualization in Science*, 1999.

[16] Y. Song, W. Chen, H. Bai, C. Lin, and E. Chang, "Parallel spectral clustering," in *ECML*, 2008.

[17] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[18] B. A. Prakash, M. Seshadri, A. Sridharan, S. Machiraju, and C. Faloutsos, "Eigenspokes: Surprising patterns and community structure in large graphs," *PAKDD*, 2010.

[19] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: Counting triangles in massive graphs with a coin," *KDD*, 2009.

[20] C. Tsourakakis, "Fast counting of triangles in large real networks without counting: Algorithms and laws," in *ICDM*, 2008.

[21] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," *SIGCOMM*, pp. 251–262, Aug-Sept. 1999.

[22] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce," *ICDM*, 2008.

[23] U. Kang, D. H. Chau, and C. Faloutsos, "Mining large graphs: Algorithms, inference, and discoveries," *IEEE International Conference on Data Engineering*, 2011.

[24] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec, "Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations," in *SDM*, 2010, pp. 548–558.

[25] C. Liu, H. chih Yang, J. Fan, L.-W. He, and Y.-M. Wang, "Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce," in *WWW*, 2010, pp. 681–690.

[26] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec, "Hadi: Mining radii of large graphs," *ACM Trans. Knowl. Discov. Data*, vol. 5, pp. 8:1–8:24, February 2011.

[27] U. Kang, C. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system - implementation and observations," *ICDM*, 2009.

[28] L. N. Trefethen and D. Bau III, "Numerical linear algebra," *SIAM*, 1997.

[29] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," *J. Res. Nat. Bur. Stand.*, 1950.

[30] G. H. Golub and C. F. V. Loan, “Matrix computations,” *Johns Hopkins University Press*, 1996.
[31] J. W. Demmel, “Applied numerical linear algebra,” *SIAM*, 1997.
[32] U. Kang, H. Tong, J. Sun, C.-Y. Lin, and C. Faloutsos, “Gbase: an efficient analysis platform for large graphs,” *The VLDB Journal*, pp. 1–14.
[33] C. Eckart and G. Young, “The approximation of one matrix by another of lower rank,” *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.
[34] H. Zha, X. He, C. Ding, M. Gu, and H. Simon, “Spectral relaxation for k-means clustering,” *Advances in Neural Information Processing Systems*, vol. 2, pp. 1057–1064, 2002.
[35] G. Golub and W. Kahan, “Calculating the singular values and pseudo-inverse of a matrix,” *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, vol. 2, no. 2, pp. pp. 205–224, 1965. [Online]. Available: http://www.jstor.org/stable/2949777
[36] C. C. Paige, “Bidiagonalization of matrices and solution of linear equations,” *SIAM Journal on Numerical Analysis*, vol. 11, no. 1, pp. pp. 197–209, 1974.
[37] R. Penrose, “A generalized inverse for matrices,” in *Proc. Cambridge Philos. Soc*, vol. 51, no. 3. Cambridge Univ Press, 1955, pp. 406–413.
[38] J. Kleinberg, “Authoritative sources in a hyperlinked environment,” in *Proc. 9th ACM-SIAM SODA*, 1998.
[39] J. Leskovec, D. Chakrabarti, J. M. Kleinberg, and C. Faloutsos, “Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication,” *PKDD*, 2005.
[40] Y. Zhao, X. Chi, and Q. Cheng, “An implementation of parallel eigenvalue computation using dual-level hybrid parallelism,” *Lecture Notes in Computer Science*, 2007.
[41] M. R. Guarracino, F. Perla, and P. Zanetti, “A parallel block lanczos algorithm and its implementation for the evaluation of some eigenvalues of large sparse symmetric matrices on multicomputers,” *Int. J. Appl. Math. Comput. Sci.*, 2006.
[42] R. B. Lehoucq, D. C. Sorensen, and C. Yang, “Arpack user’s guide: Solution of large-scale eigenvalue problems with implicitly restarted arnoldi methods,” *SIAM*, 1998.
[43] L. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, and I. Dhillon, “Scalapack users’s guide,” *SIAM*, 1997.
[44] P. Alpatov, G. Baker, C. Edward, J. Gunnels, G. Morrow, J. Overfelt, R. van de Gejin, and Y.-J. Wu, “Plapack: Parallel linear algebra package - design overview,” *SC97*, 1997.
[45] “Mahout information,” http://lucene.apache.org/mahout/.
[46] R. Lämmel, “Google’s mapreduce programming model – revisited,” *Science of Computer Programming*, vol. 70, pp. 1–30, 2008.
[47] “Hadoop information,” http://hadoop.apache.org/.
[48] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, “Pig latin: a not-so-foreign language for data processing,” in *SIGMOD ’08*, 2008.
[49] U. Kang, B. Meeder, and C. Faloutsos, “Spectral analysis for billion-scale graphs: Discoveries and implementation,” in *PAKDD (2)*, 2011, pp. 13–25.
[50] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, “Scope: easy and efficient parallel processing of massive data sets,” *VLDB*, 2008.
[51] R. L. Grossman and Y. Gu, “Data mining using high performance data clouds: experimental studies using sector and sphere,” *KDD*, 2008.
[52] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, “Interpreting the data: Parallel analysis with sawzall,” *Scientific Programming Journal*, 2005.
[53] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, “Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries,” in *KDD*, 2012, pp. 316–324.

U Kang received Ph.D. in Computer Science at Carnegie Mellon University. He received B.S. in Computer Science and Engineering at Seoul National University. He won two best paper awards. He has published over 20 refereed articles in major data mining and database venues. He holds two U.S. patents. His research interests include data mining in massive graphs.

Brendan Meeder is a PhD student in the School of Computer Science at Carnegie Mellon. After getting his B.S. in computer science from Carnegie Mellon, he worked for a year at Microsoft Research investigating mobile speech-recognizing devices. He is interested in the intersection of mathematical models of networks and large-scale data analysis of real systems.

Evangelos E. Papalexakis is a Ph.D. student in School of Computer Science, Carnegie Mellon University. He received his Electronic and Computer Engineering Diploma and MSc from the Technical University of Crete, Greece. His research interests include data mining, multiway/tensor analysis and anomaly detection in massive graphs.

Christos Faloutsos is a Professor at Carnegie Mellon University. He has received the Presidential Young Investigator Award by the National Science Foundation (1989), the Research Contributions Award in ICDM 2006, the SIGKDD Innovations Award (2010), eighteen “best paper” awards (including two “test of time” awards), and four teaching awards. He is an ACM Fellow, he has served as a member of the executive committee of SIGKDD; he has published over 200 refereed articles, 11 book chapters and one monograph. He holds six patents and he has given over 30 tutorials and over 10 invited distinguished lectures. His research interests include data mining for graphs and streams, fractals, database performance, and indexing for multimedia and bio-informatics data.