Clustered Embedding of Massive Social Networks

Han Hee Song*  Berkant Savas†  Tae Won Cho*  Vacha Dave*
Zhengdong Lu†  Inderjit S. Dhillon*†  Yin Zhang*  Lili Qiu*

April 26, 2012

The University of Texas at Austin
Department of Computer Science
Technical report TR–12–06

Abstract
The explosive growth of social networks has created numerous exciting research opportunities. A central concept in the analysis of social networks is a proximity measure, which captures the closeness or similarity between nodes in a social network. Despite much research on proximity measures, there is a lack of techniques to efficiently and accurately compute proximity measures for large-scale social networks. In this paper, we develop a novel dimensionality reduction technique, called clustered spectral graph embedding, to embed the graphs adjacency matrix into a much smaller matrix. The embedded matrix together with the embedding subspaces capture the essential clustering and spectral structure of the original graph and allows a wide range of analysis tasks to be performed in an efficient and accurate fashion. To evaluate our technique, we use three large real-world social network datasets: Flickr, LiveJournal and MySpace, with up to 2 million nodes and 90 million links. Our results clearly demonstrate the accuracy, scalability and flexibility of our approach in the context of three important social network analysis tasks: proximity estimation, missing link inference, and link prediction.

1 Introduction
1.1 Motivation
Social networks have gained tremendous popularity recently. Social networking sites, such as MySpace [41], Facebook [15], YouTube [58], Twitter [53] and LiveJournal [34] have each attracted tens of millions of visitors each month [43] and are among the most popular sites on today’s Internet [3]. The explosive growth of social networks creates exciting research opportunities in network security (e.g., fighting spam [17], defending against Sybil attacks [59, 55]), systems research (e.g., constructing socially aware overlay networks [44] and systems), information technology (e.g., improving Internet search [37] and content recommendation [5]), business (e.g., fraud detection [10], viral marketing [20]), and social sciences (e.g., modeling complex networks [4, 11]).

A central concept in the analysis of social networks is a proximity measure, which quantifies the closeness or similarity between nodes in a social network. Intuitively, socially “close” users are often more trustworthy (which is useful for fraud detection [10], spam mitigation [17], and Sybil attack defense [59, 55]), and tend to have similar interests (which is useful for improving Internet search [37] and content recommendation [5]).

*Department of Computer Science, The University of Texas at Austin.
†Institute for Computational Engineering and Sciences, The University of Texas at Austin.
A variety of effective proximity measures have been proposed, such as the number of common neighbors, the Katz measure [21], rooted PageRank [33], and escape probability [52]. Despite their effectiveness, however, many proximity measures have high computational complexity and are considered prohibitive for large social networks [48, 52].

Recently, Song et al. [51] made significant progress on scalable proximity estimation in large social networks. Their key observation is that for certain proximity measures, such as the Katz measure, while the complete proximity matrix (which specifies the proximity between all node pairs) is massive (with millions of rows and columns), it can be accurately approximated as the product of two factor matrices with much smaller size. In other words, the proximity matrix has a good low-rank approximation. Similar ideas based on low-rank approximation have also been successfully exploited in network coordinate systems to approximate delay between two arbitrary Internet hosts (e.g., [42, 36]), and in network compressive sensing to infer missing elements of a network data matrix from few measurements [60].

1.2 Challenges

Despite much progress in low-rank approximation of massive network matrices, three significant challenges remain: (i) **Accuracy.** Existing low-rank approximation techniques are only effective when the matrix of interest has comparatively low rank. For example, the proximity estimation techniques in [51] use factor matrices with rank 60. Similarly, existing network coordinate systems typically use low-rank approximation with rank below 20. Unfortunately, given the massive scale and enormous complexity of social networks, so few dimensions may not capture enough information about the underlying social structure. Indeed, as shown in [51], proximity measures, such as rooted PageRank [33] and escape probability [52], have much higher intrinsic dimensionality and thus cannot be approximated accurately by existing techniques. (ii) **Scalability.** Existing techniques require computing a large number of rows and columns of the proximity matrix (also known as “landmarks”) in order to derive the low-rank approximation. Such preprocessing becomes increasingly expensive. Our results in Section 5.2 suggest that with just 2.5% of MySpace’s user accounts, the preprocessing already takes over 5 hours. Further scaling up the method poses significant scalability challenges as the size of the network grows. (iii) **Flexibility.** As shown in [51], no proximity measure performs consistently well across different social networks. Their effectiveness is also sensitive to the choice of control parameters, which is difficult to tune for unclear network characteristics each social graph has. It is desirable to have a flexible technique that automatically learns the optimal parameter configuration and performs consistently better than other measures.

1.3 Approach and contributions

To address the above challenges, we develop a novel dimensionality technique, called **clustered spectral graph embedding,** which embeds the original highly sparse but massive social graph into a dense but much smaller matrix representation. The embedded matrix together with the embedding subspaces capture the essential clustering and spectral structure of the original graph, and allows a wide range of analysis tasks to be performed in an efficient and accurate fashion. The technique is also flexible, and can naturally support incremental update and parallel/distributed computation, which is essential for social networks that are highly dynamic and/or decentralized.

We use three large real-world social network datasets (LiveJournal [34], Flickr [16] and MySpace [41] with up to 2 million nodes and 90 million links) to experimentally demonstrate the effectiveness of our approach. In particular, we evaluate our methods accuracy, scalability and flexibility in the context of three important social network analysis tasks: (i) **proximity estimation** (i.e., approximating well-known proximity measures proposed in the literature), (ii) **missing link inference** (i.e., inferring the locations of unobserved links based on observed links), and (iii) **link prediction** (i.e., predicting which node pairs will become connected based on past snapshots of the social network).

In the context of proximity estimation, our new techniques result in nearly an order of magnitude speedup over the state-of-the-art proximity estimation techniques [51]. More importantly, with the same memory requirement, our technique is able to create approximations when the rank is an order of magnitude higher
than previous methods. As a result, our technique results in dramatic improvement on the approximation accuracy of proximity measures, such as rooted PageRank and escape probability, which are not low-rank and thus cannot be approximated accurately by previous methods. In the context of missing link inference, our technique results in several-fold reduction in the false positive rate subject to the same false negative rate. In the context of link prediction, our technique yields a novel supervised proximity measure that significantly improves link prediction accuracy. These results clearly demonstrate the effectiveness of our approach.

1.4 Organization
The remainder of the paper is organized as follows. In Section 2, we provide background on proximity measures, spectral embedding and clustering. In Section 3, we present details on our clustered spectral graph embedding approach. In Section 4, we explore three important applications: proximity estimation, missing link inference, and link prediction, and then evaluate the effectiveness of our methods on these applications in Section 5. In Section 6, we survey the related work, and conclude in Section 7.

2 Preliminaries
A social network can naturally model the explicit friendship or trust relationship among users in a social networking site like Facebook [15]. A social network can also be defined implicitly to quantify the interaction levels among users. For example, one can construct a social network based on the past (non-spam) email exchanges between users [17], or the traffic volumes transmitted between Internet hosts.

Formally, we denote a social network as a graph \( G = (V, E) \), where \( V = \{1, 2, \ldots, |V|\} \) is a set of vertices, and \( E = \{e_{ij} \mid i, j \in V\} \) is a set of edges. In particular, if there is an edge between vertex \( i \) and vertex \( j \), then \( e_{ij} \) denotes the weight of this edge. The adjacency matrix \( A \) of the graph \( G \) is an \( m \times m \) matrix with \( m = |V| \):

\[
a_{ij} = A[i, j] = \begin{cases} e_{ij}, & \text{if there is an edge between } i \text{ and } j, \\ 0, & \text{otherwise.} \end{cases}
\]

Below we first formally define some of the most commonly used proximity measures, all of which can benefit from our new dimensionality reduction technique. We then introduce three important concepts: (i) graph embedding, (ii) spectral graph embedding, and (iii) graph clustering. These concepts are essential for understanding our technique, which is a novel graph embedding algorithm that effectively combines graph clustering and spectral graph embedding.

2.1 Proximity measures
Proximity measures are important for many social network applications. Most proposed proximity measures can be divided into two broad categories: (1) direct measures that are based on shortest graph distances or the maximum information flow between two nodes or node neighborhood, e.g., common neighbors; and (2) more sophisticated measures that include infinite sums over ensembles of all paths between two nodes, e.g., Katz measure [21], rooted PageRank [33], and escape probability [52]. It has been shown that the path-ensemble based proximity measures capture much more information about the underlying network compared to the direct measures and are generally more effective for various tasks [33, 52]. Now, we define the proximity measures mentioned above.

2.1.1 Common neighbors
Let \( N_i \) be the neighbor set of vertex \( i \). Then, the common neighbors proximity measure is

\[
P_{cn}[i, j] = |N_i \cap N_j|.
\]

If the number of common neighbors is high between vertices \( i \) and \( j \), then it is more likely they will get connected. For an unweighted graph (i.e., \( e_{ij} = 1 \) for all edges), the common neighbor for all vertex pairs is simply given by \( P_{cn} = A^2 \).
2.1.2 The Katz measure

Let $p_{ij}^{(k)}$ denote the number of paths of length $k$ between vertices $i$ and $j$. Then, the Katz measure is

$$P_{kz}[i,j] = \sum_{k=1}^{\infty} \beta^k p_{ij}^{(k)},$$

where $\beta$ is a damping parameter. A high value of the Katz measure between two vertices signifies a stronger relationship. Using the adjacency matrix $A$, we may write the Katz measure for all vertex pairs simultaneously as

$$P_{kz} = \sum_{k=1}^{\infty} \beta^k A^k = (I - \beta A)^{-1} - I,$$

where $I$ is an identity matrix, and $\beta < 1/\|A\|$. 

2.1.3 Rooted PageRank

The rooted PageRank measure (rooted at vertex $i$) is the probability of landing at vertex $j$ in a random walk with a probability $\alpha$ of jumping to vertex $i$ in each step, and with a probability $1 - \alpha$ that the process continues to a random neighbor. Let $D$ be the diagonal degree matrix given by $D[i,i] = \sum_j A[i,j]$. Let $T = D^{-1/2}AD^{-1/2}$ be the normalized adjacency matrix. The stationary probability of the rooted PageRank for all vertex pairs is given by

$$P_{rpr} = (1 - \alpha)(I - \alpha D^{-1}A)^{-1} = (1 - \alpha)D^{-1/2} \left( \sum_{k=0}^{\infty} \alpha^k T^k \right) D^{1/2}.$$

2.1.4 Escape probability

The escape probability $P_{ep}[i,j]$ from vertex $i$ to $j$ is the probability that a random walker (who started at $i$) will visit $j$ before returning back to $i$. With $\alpha$ being the random jump probability, the measure is derivable from rooted PageRank by

$$P_{ep}[i,j] = f(P_{rpr}[i,j]),$$

where the function $f$ is defined as

$$f(P, i, j) = \frac{(1 - \alpha)P[i,j]}{P[i,i]P[j,j] - P[i,j]P[j,i]}.$$ (1)

2.2 Graph embedding

Let $A$ be an $m \times m$ adjacency matrix of a graph. For simplicity, we assume that $A$ is symmetric. In section 3.3 we will show how to extend the presented concepts to non-symmetric adjacency matrices. A graph embedding can be mathematically formalized as the decomposition

$$A_{m \times m} \approx U_{m \times r}L_{r \times r}U_{m \times r}^T = ULU^T,$$ (2)

where $U$ is an $m \times r$ orthonormal matrix (i.e., $U^T U = I_r$ is an identity matrix), and $L$ is an $r \times r$ matrix. $U$ represents a basis for the embedding subspace and $L$ represents the embedded adjacency matrix of the original graph. Since $U$ is orthonormal, (Eq. 2) can be applied to approximate any matrix power $A^k$ by

$$A^k \approx (ULU^T)^k = UL^kU^T.$$ (3)

As a special case, with $k = 2$, we get the frequently used common neighbor proximity measure $P_{cn} = A^2 \approx UL^2U^T$. Many functions of $A$ can be approximated using a sum of matrix powers through the Taylor series.
expansion. Using (Eq. 3) we can approximate these functions with corresponding functions on $L$. This can significantly reduce the computational cost as typically $r \ll m$ and most of the calculations will involve the $r \times r$ matrix $L$ instead of the $m \times m$ matrix $A$. For example, using (Eq. 2) and (Eq. 3), we can approximate the Katz measure as

$$P_{Kz} \approx \sum_{k=1}^{\infty} \beta^k U L^k U^T = U ((I_r - \beta L)^{-1} - I_r) U^T.$$  

2.3 Spectral graph embedding

The best rank-$r$ approximation of $A$, in terms of squared approximation error, is given by the $r$-dimensional spectral graph embedding (SGE):

$$A \approx U \Lambda U^T,$$ (4)

where $\Lambda$ is a diagonal matrix with the $r$ largest (in magnitude) eigenvalues of $A$, and $U$ contains the corresponding eigenvectors. Figure 2(a) shows a pictorial illustration of the spectral graph embedding.

Over the past several decades, eigendecomposition and spectral graph embedding have been important research tools for achieving dimensionality reduction on large matrices and graphs. Although there are computationally efficient algorithms [28, 27] to compute the spectral embedding of large sparse matrices, they are still quite expensive on massive social networks with millions of nodes. As a result, spectral graph embedding can only afford to work with relatively small $r$, which may not capture sufficient social/network structure for very large social networks and thus yields poor approximation accuracy (see Section 5).

2.4 Graph clustering

A key step in the methods we will propose is to cluster or partition a graph. Given a graph $G = (V, E)$ there are various objective functions that measure the quality of the clustering, e.g., ratio cut [19] and normalized cut [50]. Although these graph clustering objectives are NP-hard to optimize [56], there are several efficient clustering algorithms that often produce good quality results, e.g., GRACLUS [12], METIS [1], and modularity optimization [6].

Assume that we have a clustering of $G(V, E)$ into $c$ disjoint clusters specified by the vertex sets $V_i$, $i = 1, \ldots, c$, i.e., $\bigcup_{i=1}^{c} V_i = V$ and $V_i \cap V_j = \emptyset$ for all $i \neq j$. Let $m_i = |V_i|$. Without loss of generality, we can assume that the vertices in $V_1, \ldots, V_c$ are in a strictly increasing order. Then the adjacency matrix $A$ will have the following form

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & A_{cc} \end{bmatrix},$$ (5)

where each diagonal block $A_{ii}$ is an $m_i \times m_i$ matrix, that can be considered as a local adjacency matrix for cluster $i$. The off-diagonal blocks $A_{ij}$ ($i \neq j$) are $m_i \times m_j$ matrices that contain the set of edges incident on vertices belonging to different clusters. In an ideal scenario, with perfect clustering, the off-diagonal blocks will not contain any edges, thus $A_{ij} = 0$, and the graph will have $c$ connected components. On the other hand, in the more general scenario $A$ is comprised of a single connected component with a clear clustering structure, a naive approximation of the original graph could be obtained through its diagonal blocks $A \approx \text{diag}(A_{11}, \ldots, A_{cc})$. By further introducing low-rank approximations $A_{ii} \approx V_i \Lambda_i V_i^T$, we have

$$A \approx \text{diag}(V_1 \Lambda_1 V_1^T, \ldots, V_c \Lambda_c V_c^T).$$

3 Clustered Spectral Graph Embedding

3.1 Basic algorithm

Our proposed method, clustered spectral graph embedding (CSGE) improves the efficiency and accuracy of approximating various proximity measures by effectively combining clustering with spectral graph embed-
Figure 1: Sparsity pattern of an adjacency matrix with $c = 10$ clusters; 80% of the edges are within the diagonal blocks $A_{ii}$.

Recall that $A$ is the adjacency matrix of a graph. We will assume that the graph has been partitioned into $c$ clusters and the vertices are ordered so that the diagonal blocks $A_{ii}$, $i = 1, \ldots, c$, correspond to the local adjacency matrices for the different clusters as in (Eq. 5). Since the block partitioning of $A$ is obtained through clustering of the graph, it follows that most of the edges are within the diagonal blocks. Only a small fraction of the edges are between clusters and are consequently located in the off-diagonal blocks. Figure 1 shows a typical sparsity pattern of an adjacency matrix after a clustering step. In this particular case, we have clustered the graph into $c = 10$ clusters and 80% of the edges are within the diagonal blocks.

Computing the best rank-$r_i$ approximations through spectral graph embedding for every cluster (diagonal block), we get

$$A_{ii} \approx V_i \Lambda_i V_i^T,$$

where $\Lambda_i$ is a diagonal matrix and contains the $r_i$ largest eigenvalues of $A_{ii}$ (in magnitude), and $V_i$ is an orthonormal matrix with the corresponding eigenvectors. Due to the orthonormality of $V_i$ it follows that the matrix

$$V = \text{diag}(V_1, \ldots, V_c)$$

is also orthonormal. We can now use this block-diagonal matrix $V$ to obtain a graph embedding for the entire adjacency matrix $A$. The graph embedding may be written as $A \approx VS V^T$. Since $V$ is orthonormal, it can be shown that the optimal $S_i$ in least squares sense, is

$$S = V^T A V = \begin{bmatrix} V_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & V_c & \cdots & A_{cc} \end{bmatrix} \begin{bmatrix} A_{11} & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & A_{cc} \end{bmatrix} \begin{bmatrix} V_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & V_c & \cdots & V_c \end{bmatrix} = \begin{bmatrix} S_{11} & \cdots & S_{1c} \\ \vdots & \ddots & \vdots \\ S_{c1} & \cdots & S_{cc} \end{bmatrix},$$

where $S_{ij} = V_i^T A_{ij} V_j$, for $i,j = 1, \ldots, c$. Using (Eq. 6) we can verify that $S_{ii} = \Lambda_i$ are diagonal. The off-diagonal blocks $S_{ij}$, on the other hand, capture interactions between different clusters. We obtain the following approximation

$$A \approx VS V^T = \begin{bmatrix} V_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & V_c & \cdots & S_{cc} \end{bmatrix} \begin{bmatrix} S_{11} & \cdots & S_{1c} \\ \vdots & \ddots & \vdots \\ S_{c1} & \cdots & S_{cc} \end{bmatrix} \begin{bmatrix} V_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & V_c & \cdots & V_c \end{bmatrix}^T,$$

$$A \approx VS V^T.$$
which we call *clustered spectral graph embedding* (CSGE). For example, with $c = 3$ we can write

$$
A \approx \begin{bmatrix}
V_1 & 0 & 0 \\
0 & V_2 & 0 \\
0 & 0 & V_3
\end{bmatrix}
\begin{bmatrix}
S_{11} & S_{12} & S_{13} \\
S_{21} & S_{22} & S_{23} \\
S_{31} & S_{32} & S_{33}
\end{bmatrix}
\begin{bmatrix}
V_1 & 0 & 0 \\
0 & V_2 & 0 \\
0 & 0 & V_3
\end{bmatrix}^T.
$$

Figure 2 shows an illustration of the SGE compared to the CSGE. In analogy to the terminology as spectral graph embedding, we denote $V = \text{diag}(V_1, \ldots, V_c)$ as a basis for the *clustered embedding subspace*, and $S$ as the *embedded adjacency matrix* for CSGE. Note that with $c = 1$, CSGE becomes the regular SGE.

In related preliminary evaluation, we explored the benefit of combining clustering with different local (within-cluster) low rank approximation schemes from the viewpoint of numerical accuracy on a few static graphs [49]. However, higher numerical accuracy in matrix approximations does not necessarily translate to benefits in end applications, such as link prediction. Additionally, social network analysis requires more than matrix approximation in order to accommodate scalability and flexibility for various algorithms. In the subsequent sections, we develop and validate CSGE in the context of large, dynamic social networks.

### 3.2 Advantages

Compared with the rank-$r$ SGE $A \approx UAV^T$, the CSGE $A \approx VSV^T$ from (Eq. 7) can achieve much higher accuracy while using a comparable amount of memory. In many cases CSGE is also computationally more efficient. To simplify the discussion for the CSGE we will use $r$-dimensional graph embeddings for each cluster $A_{ii}$ (i.e., $r_i = r$).

#### 3.2.1 Efficiency in computation and memory usage

From the block-diagonal structure of $V$, we immediately observe that we have a $cr$-dimensional embedding in the CSGE, while only an $r$-dimensional embedding in the SGE. Thus, there are $c$ times more columns in $V_{m \times cr}$ than in $U_{m \times r}$. However, $V_{m \times cr}$ has exactly the same $O(mr)$ memory usage as $U_{m \times r}$, since we only store the diagonal blocks $V_i$, and not the off-diagonal blocks, which are all zeros (see Figure 2). Comparing the embedded adjacency matrices $S$ from the CSGE and $\Lambda$ from the SGE, we see that the $cr \times cr$ matrix $S$ is larger and dense, while the $\Lambda$ is $r \times r$ and diagonal. Therefore, CSGE uses $O(mr + c^2r^2)$ total memory and SGE uses $O(mr + r)$. For large graphs with millions of vertices, the memory complexity is dominated by $U$ or $V$ because we have $m \gg r$ and $m \gg cr$. For example, in the LiveJournal dataset $m = 1,770,961$. With (typical values) $r = 100$ and $c = 50$, $S$ accounts for only 12% of the total memory usage of the CSGE. As shown in Section 3.3, we can further reduce the memory usage of $S$ through another embedding. So the total memory usage of the CSGE is comparable to that of SGE.
With state-of-the-art algorithms for computing a small number of eigenvectors of a large and sparse matrix (e.g., [28, 29]), the SGE has a memory complexity of $O(m(r + p))$, where $p$ is a user specified parameter and usually $p \approx r$. Smaller $p$ requires less memory but the convergence becomes slower, and vice versa. For the CSGE, the approximation of the different clusters is independent of each other. These computations can be done using the entire memory available. Since the size of each cluster $m_i = |V_i|$ may be orders of magnitude smaller than $m = \sum_{i=1}^{c} m_i$, we can compute the per-cluster spectral graph embedding with a much larger dimension than otherwise possible. For example, when $c = 10$, the average cluster size would be about $m/10$. Allowing the CSGE to fully utilize the entire memory space would increase the maximum computable dimensions from $r$ to $10r$.

The time complexity for computing the CSGE or the SGE is dominated by the cost of the eigendecomposition. State-of-the-art algorithms (e.g., [28, 29]) for sparse eigendecomposition are typically iterative algorithms. The time complexity for each iteration scales linearly with respect to the number of non-zeros in the input matrix, whereas the number of iterations required to achieve convergence depends on the gap between adjacent eigenvalues. For example, it typically takes many more iterations to decompose the normalized adjacency matrix $T = D^{-1/2}AD^{-1/2}$ than the original adjacency matrix $A$ because eigenvalues of $T$ have smaller gaps.

It is easy to see that the number of non-zeros in the global adjacency matrix $A$ is larger than the total number of non-zeros in all the per-cluster adjacency matrices $A_{ii}$. Hence, the per-iteration cost for decomposing $A$ is higher than the total per-iteration cost for decomposing all the $A_{ii}$. In addition, our experience suggests that it often takes a larger number of iterations for the global eigendecomposition to converge. As a result, it is often much faster to compute many $r$-dimensional graph embeddings of smaller matrices $A_{ii}$ than a single $r$-dimensional graph embedding computation of a large $A$. Our experiments results in Section 5 show that even after including the initial clustering time of the graph as well as the computation time for $S$, the CSGE is still much faster than the SGE. In the specific case when the normalized adjacency matrix $T$ is used, CSGE is an order of magnitude faster than SGE. Moreover, with parallel/distributed computation of clusters detailed in Section 3.3, we can further improve the timing efficiency of CSGE.

### 3.2.2 Accuracy

An important consequence of explicitly focusing on each cluster of the network is that using the same amount of memory the CSGE yields a significantly smaller residual than the residual for the regular SGE, i.e.,

$$\|A - VSV^T\|_F < \|A - U\Lambda U^T\|_F,$$

where $\|X\|_F = \sqrt{\sum_{i,j} X[i,j]^2}$ is the Frobenius norm of $X$. Recall that the SGE is optimal with respect to the dimension of the embedding (or with respect to the rank in the approximation), but in terms of memory consumption the CSGE gives significantly better embedding as discussed in [49] and Section 5. The accuracy benefit of CSGE is most significant when we are able to transform (using permutations) a given adjacency matrix so that most of the non-zeros are contained within the diagonal blocks. This property is closely related to a graph forming good clusters. Many (if not most) real-world graphs and social networks exhibit this property of forming reasonably good clusters [30, 31]. This is certainly the case for the datasets we have considered in this paper.

Interestingly, even if the original graph does not form good clusters or if the clustering algorithm performs poorly, clustered graph embedding can still achieve lower residual error than the regular SGE. The following theorem establishes this guarantee for the special case with $c = 2$ clusters.

**Theorem 1.** Let $A \approx U\Lambda U^T$ be the $r$-dimensional SGE. Split $U$ into any two parts $U^T = [U_1^T U_2^T]$. Let $U_i = Q_iR_i$ be the QR decomposition [18] of $U_i$, where $Q_i$ is orthonormal (i.e., $Q_i^TQ_i = I_r$) and $R_i$ is upper triangular. Let $V = \text{diag}(Q_1, Q_2)$ and let $S = V^TAV$. We have

$$\|A - VSV^T\|_F \leq \|A - U\Lambda U^T\|_F.$$
Proof. Through simple matrix calculation, we have:

$$UΛU^T = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} Λ & Λ \\ Λ & Λ \end{bmatrix} \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix}^T = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \begin{bmatrix} R_1ΛR_1^T & R_1ΛR_2^T \\ R_2ΛR_1^T & R_2ΛR_2^T \end{bmatrix} \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix}^T = VΛV^T.$$  

It is evident that $V$ is orthonormal. So $S = V^TAV$ is an optimal solution to $\min_S \|A - VS\|_F$. As a result, we have $\|A - VS\|_F \leq \|A - VΛV^T\|_F = \|A - UΛU^T\|_F$.  

The above theorem can be easily generalized to the case with $c > 2$ clusters. Therefore, under any arbitrary clustering of the original graph, there always exists a clustered graph embedding that has lower residual error than the regular SGE.

3.3 Extensions

The basic CSGE algorithm can be further extended to support (i) asymmetric adjacency matrices (directed graphs), (ii) further embedding of the embedded adjacency matrix, (iii) incremental updates, and (iv) parallel/distributed computation.

3.3.1 Asymmetric adjacency matrices

There are two possible strategies to cope with an asymmetric adjacency matrix $A$:

(1) When the fraction of asymmetric edges (i.e., vertex pairs $(i, j)$ such that $A[i, j] \neq A[j, i]$) is not high, we can simply derive the basis of the clustered embedding subspace (i.e., the $V$ matrix) using a symmetrized version of $A$, e.g., $A_{sym} = \frac{1}{2}(A + A^T)$ or $A_{sym} = \max(A, A^T)$. We can then capture the asymmetry of $A$ by solving $\min_S \|A - VS\|_F$, yielding $S = V^TAV$.

(2) Alternatively, we can apply the singular value decomposition (SVD) [18] to compute the best rank-$r_i$ approximation for every cluster (diagonal block), yielding

$$A_{ii} \approx U_iΣ_iV_i^T, \quad i = 1, \ldots, c,$$

where $Σ_i$ is a diagonal matrix containing the $r_i$ largest singular values of $A_{ii}$, $U_i$ and $V_i$ are orthonormal matrices with the corresponding left and right singular vectors, respectively. Due to the orthonormality of $U_i$ and $V_i$, $U \hat{=} \text{diag}(U_1, \ldots, U_c)$ and $V \hat{=} \text{diag}(V_1, \ldots, V_c)$ are also orthonormal. We can then use the block-diagonal matrices $U$ and $V$ to obtain an asymmetric clustered graph embedding for the entire adjacency matrix $A$ written as $A \approx USV^T$. The optimal $S$, in least squares sense, is

$$S = U^TAV = \begin{bmatrix} S_{11} & \cdots & S_{1c} \\ \vdots & \ddots & \vdots \\ S_{c1} & \cdots & S_{cc} \end{bmatrix},$$

where $S_{ij} = U_i^TA_{ij}V_j$, for $i, j = 1, \ldots, c$. The asymmetric graph embedding $A \approx USV^T$ has properties very similar to the symmetric graph embedding. For example, it can be applied to efficiently approximate matrix powers $A^k \approx (USV^T)^k = U(SV^TU)^{k-1}SV^T$.

3.3.2 Further embedding

With a large number of clusters, and larger ranks in the approximations, the size of $S$ could become too large. To reduce memory usage, we can further compute the SGE of the embedded adjacency matrix $S$. That is, we further approximate $S$ by keeping the largest eigenvalues (in magnitude), i.e., $S \approx QΛQ^T$, where $Λ$ contains the largest eigenvalues of $S$ and $Q$ contains the corresponding eigenvectors. The combined
approximation is then \( A \approx V \bar{Q} \bar{\Lambda} \bar{Q}^T \). We would like to remark that it is possible to compute \( \bar{Q} \) and \( \bar{\Lambda} \) without having to explicitly compute the dense matrix \( S \). All we need is to treat \( S = V^T AV \) as an operator acting on a vector. Then, for a given vector \( v \), we compute \( Sv = (V^T AV)v = V^T(A(Vv)) \). Since \( V \) is block diagonal, and \( A \) is sparse, computing matrix-vector products with both \( V \) and \( A \) is efficient. State-of-the-art algorithms for sparse eigendecomposition (e.g., [28, 29]) can then efficiently compute a SGE \( S \approx \bar{Q} \bar{\Lambda} \bar{Q}^T \) using such matrix-vector products.

3.3.3 Incremental updates

As many social networks are highly dynamic, it is desirable to cheaply update the graph embedding equation \( A \approx VSV^T \) under the new adjacency matrix \( A' = A + \Delta_A \) when \( \Delta_A \) has only few non-zero elements. A simple strategy is to fix \( V \) and only update \( S \) with

\[
S' = V^T A' V = V^T (A + \Delta_A) V = S + V^T \Delta_A V = S + \Delta_S,
\]

where \( \Delta_S = V^T \Delta_A V \) contains the updates to \( S \) and can be efficiently computed due to the sparsity of \( \Delta_A \). The embedding for the new adjacency matrix \( A' \) is then \( A' \approx VS'V^T \).

3.3.4 Parallel/distributed computation

CSGE is naturally suited for parallel/distributed computation. For example, consider the email exchange graph of a large corporation, which may consist of a number of organizations. Instead of requiring a separate clustering step, we can directly partition users based on their natural organizational boundaries. Each organization (i.e., cluster) \( i \) can then derive \( V_i \) locally based on its internal email exchange subgraph \( A_{ii} \) according to \( A_{ii} \approx V_i \Lambda_i V_i^T \). Each pair of organizations \( i \) and \( j \) can jointly compute the block \( S_{ij} \) of the embedded adjacency matrix \( S \) based on their \( V_i, V_j \) and the inter-organization email exchange subgraph \( A_{ij} \), which can again be locally monitored. The ability to support parallel/distributed computation can further improve efficiency in parallel/distributed computational environments.

4 Application to social networks

In this section, we show how to utilize CSGE and SGE in three important social network analysis tasks: proximity estimation, missing link inference, and link prediction.

4.1 Proximity estimation

In Section 2.1 we defined four proximity measures: common neighbors, Katz measure, rooted PageRank, and escape probability. Each of these proximity measures will be computed or approximated using three different approaches: (i) direct computation; (ii) approximation using the spectral graph embedding \( A \approx U\Lambda U^T \) from (Eq. 4); and (iii) approximation using the clustered spectral graph embedding \( A \approx VSV^T \) from (Eq. 7). We obtain 12 different proximity measures as summarized in Table 1. Note that following [51], the summation for the Katz measures are truncated to only include \( k_{\text{max}} = 6 \) terms, i.e., at most length-\( k_{\text{max}} \) paths are taken into account. The truncation is necessary in order to compute the real proximity measures that do not use any graph embedding [51]; these serve as the ground truth for accuracy evaluation. In the rooted PageRank measures, we truncate the summation to only include \( k_{\text{max}} = 20 \) terms and we have \( T \approx D^{-1/2} AD^{-1/2} \approx U\Lambda U^T \) (for SGE) and \( T \approx VS V^T \) (for CSGE). Finally, the escape probability measures are computed from the rooted PageRank measures using the function \( f \) defined in (Eq. 1).

4.2 Missing link inference

Missing link inference aims to infer additional links that, while not directly visible, are likely to exist (based on the set of links that are directly observed). Missing link inference falls into the general realm of compressive
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Method name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cn</td>
<td>Common neighbor</td>
<td>$P_{cn} = A^T$</td>
</tr>
<tr>
<td>cn-sge</td>
<td>Common neighbor with SGE</td>
<td>$P_{cn-sge} = U\Lambda^2U^T$</td>
</tr>
<tr>
<td>cn-csge</td>
<td>Common neighbor with CSGE</td>
<td>$P_{cn-csge} = V S^2V^T$</td>
</tr>
<tr>
<td>kz</td>
<td>Katz measure</td>
<td>$P_{kz} = \sum_{k=1}^{\infty} \beta^k A^k$</td>
</tr>
<tr>
<td>kz-sge</td>
<td>Katz measure with SGE</td>
<td>$P_{kz-sge} = U \left( \sum_{k=1}^{\infty} \beta^k \Lambda^k \right) U^T$</td>
</tr>
<tr>
<td>kz-csge</td>
<td>Katz measure with CSGE</td>
<td>$P_{kz-csge} = V \left( \sum_{k=1}^{\infty} \beta^k S^k \right) V^T$</td>
</tr>
<tr>
<td>rpr</td>
<td>Rooted PageRank</td>
<td>$P_{rpr} = (1-\alpha)D^{-1/2}(\sum_{k=0}^{\infty} \alpha^k T^k) D^{1/2}$</td>
</tr>
<tr>
<td>rpr-sge</td>
<td>Rooted PageRank with SGE</td>
<td>$P_{rpr-sge} = (1-\alpha)D^{-1/2}(I + \bar{U} \left( \sum_{k=1}^{\infty} \alpha^k \bar{\Lambda}^k \right) \bar{U}^T) D^{1/2}$</td>
</tr>
<tr>
<td>rpr-csge</td>
<td>Rooted PageRank with CSGE</td>
<td>$P_{rpr-csge} = (1-\alpha)D^{-1/2}(I + \bar{V} \left( \sum_{k=1}^{\infty} \alpha^k \bar{S}^k \right) \bar{V}^T) D^{1/2}$</td>
</tr>
<tr>
<td>ep</td>
<td>Escape probability</td>
<td>$P_{ep}[i,j] = f(P_{rpr}, i, j)$</td>
</tr>
<tr>
<td>ep-sge</td>
<td>Escape probability with SGE</td>
<td>$P_{ep-sge}[i,j] = f(P_{rpr-sge}, i, j)$</td>
</tr>
<tr>
<td>ep-csge</td>
<td>Escape probability with CSGE</td>
<td>$P_{ep-csge}[i,j] = f(P_{rpr-csge}, i, j)$</td>
</tr>
</tbody>
</table>

Table 1: Description of the proximity measures.

sensing, which aims to reconstruct missing data based on indirect, partial observations. Compressive sensing has many applications both in networking [60] and beyond, and has attracted considerable research attention recently (e.g., [13, 46, 7, 60]).

4.2.1 Problem definition

Let $G = (V, E)$ be a social network with a binary adjacency matrix $A$. Let $\mathcal{O}$ be the set of observed edges or links and $\mathcal{M} = E \setminus \mathcal{O}$ be the set of missing or unobserved links. Let $A_{\mathcal{O}}$ be the incomplete adjacency matrix containing the set of observed links. $A_{\mathcal{O}}$ is defined as

$$A_{\mathcal{O}}[i,j] = \begin{cases} A[i,j], & \text{if } (i,j) \in \mathcal{O}; \\ 0, & \text{otherwise}. \end{cases}$$

Let $A_{\mathcal{M}} = A - A_{\mathcal{O}}$. Given $A_{\mathcal{O}}$, the goal of missing link inference is to infer non-zeros in $A_{\mathcal{M}}$ as they correspond to the set of missing links in $\mathcal{M}$.

4.2.2 Inference algorithm

Despite much progress on compressive sensing, we are not aware of any existing compressive sensing algorithm that can scale to massive social networks with millions of vertices. Therefore, in this paper we explore the following simple but much more scalable heuristic. We first compute proximity measures based on the observed incomplete adjacency matrix $A_{\mathcal{O}}$ (e.g., those given in Section 4.1), and then assume that the high values in these proximity measures will correspond to the set of missing links, i.e., non-zeros in $A_{\mathcal{M}}$. The threshold for determining high proximity measure values can be varied to achieve different tradeoffs between false positives and false negatives.

4.3 Link prediction

Link prediction [33] refers to the task of predicting which vertex pairs in a social network will become connected. An accurate link predictor can provide valuable insights for constructing meaningful network evolution models [4, 11]. It also allows social networks to make high-quality recommendations on potential new friends, making it easier for individual users to expand their social neighborhood. Link prediction also has direct applications in cyber security. For example, it allows one to conjecture that certain individuals in a terrorist network are working together even though their interaction has not been directly observed [24]. In this section, we show how the CSGE and supervised learning can facilitate accurate link prediction.
4.3.1 Problem definition

A natural setting for evolving social networks, or evolving networks in general, is to introduce discrete time steps \( t = 1, \ldots, t_{\text{max}} \) at which “snapshots” \( G_t = (V_t, E_t) \) of the graph are taken. Denote the corresponding adjacency matrices with \( A_1, \ldots, A_{t_{\text{max}}} \). To ease the analysis and notation, we will restrict ourselves and only use \( V_1 \) for all time steps, i.e., \( V_k = V_1 \cong V \). It is clear that the evolution of the graph is incremental. In terms of the adjacency matrices, we can express this as \( A_{t+1} = A_t + \Delta_t \), where \( \Delta_t \) contains the edges or links that are formed between time \( t \) and \( t+1 \). For each time step \( t \) we associate three sets of edges: (1) the set of existing edges \( E_t \); (2) a positive set \( N_t \) containing vertex pairs that form new links in the time interval \((t, t+1)\); and (3) a set \( Z_t \) containing vertex pairs without edges at time step \( t+1 \). Using adjacency matrices \( A_t \) and \( A_{t+1} \), we can write

\[
E_t = \{(i, j) \mid A_t[i, j] \neq 0\},
\]

\[
N_t = \{(i, j) \mid A_t[i, j] = 0 \text{ and } A_{t+1}[i, j] \neq 0\},
\]

\[
Z_t = \{(i, j) \mid A_t[i, j] = 0 \text{ and } A_{t+1}[i, j] = 0\}.
\]

It is straightforward to see that \( E_{t+1} = E_t \cup N_t \), and that the three sets \( E_t, N_t, Z_t \) are mutually disjoint.

In the link prediction problem, given \( A_t \) (or with additional previous snapshots), we try to determine \( N_t \) or equivalently find the non-zeros in \( \Delta_t \) as they correspond to newly formed links. The standard heuristic is to first compute some proximity measures based on \( A_t \) (e.g., those given in Section 4.1) and then assume that the high scores in these proximity measures will correspond to new links. We will now present several supervised proximity measures that explicitly target \( \Delta_t \) and construct graph specific models for link prediction.

4.3.2 Supervised link prediction models and framework

The link prediction models we propose have the generic prediction scores, obtained from a low dimensional graph embedding

\[
P_x = W_x F_x(x) W_x^T,
\]

where \( W_x \) represents the basis for a graph embedding, and \( F_x(x) \) is a small matrix with the model parameters \( x \) that will be learned. In particular, we will consider the following four models that are characterized by the particular form of the embedded adjacency matrix \( F_x(x) \) and the kind of graph embedding (spectral or clustered spectral) that is used.

1. **Spectral learning.** In this model, we set \( W_{\text{sGE}} = U_t \), where \( A_t \approx U_t \Lambda_t U_t^T \) is the \( r \)-dimensional SGE of the adjacency matrix \( A_t \). We then let the parameter matrix be of the form

\[
F_{\text{sGE}}(x) = \text{diag}(x_1, \ldots, x_r).
\]

2. **Clustered spectral learning.** Here we set \( W_{\text{cSGE}} = V_t \), where \( A_t \approx V_t S_t V_t^T \) is the CSGE of the adjacency matrix \( A_t \) as in (Eq. 7). The parameter matrix has the form

\[
F_{\text{cSGE}}(x) = Q_t \text{diag}(x_1, \ldots, x_{cr}) Q_t^T,
\]

where orthogonal matrix \( Q_t \) is obtained from the full eigendecomposition \( S_t = Q_t \Lambda_t S_t Q_t^T \). Recall that \( V_t \) is block diagonal and \( S_t \) is a dense matrix that captures interactions between all clusters. The number of parameters to be learned is \( cr \), where \( c \) is the number of clusters and \( r \) is the embedding dimension for each cluster.

3. **Polynomial learning.** This model is very closely related to the Katz measure. Consider \( P_{\text{KZ-SGE}} \) from Table 1. Relaxing the single parameter \( \beta \) with a different \( x_i \) for each power term we obtain a polynomial of degree \( p \). Using the SGE \( A_t \approx U_t \Lambda_t U_t^T \) and paths of length up to \( p \), the parameter matrix takes the form

\[
F_{\text{pl-sge}}(x) = \sum_{i=1}^{p} x^i A^i_t.
\]
4.3.3 Learning model parameters

We will exemplify the parameter learning step by using three snapshots, $A_1$, $A_2$, and $A_3$. We learn the parameter vector $x$ by taking $A_2$ and targeting newly formed links in $A_2$. Once the model parameters are learned we use the score function based on $A_2$ to make prediction of new links in $A_3$. Extending to more than three snapshots is straightforward. To learn the parameter matrix $F_*(x)$ we solve the following least squares problem:

$$
\min_x \sum_{(i,j) \in S} \left( [W_* F_*(x) W_*^T]_{i,j} - \Delta_1[i,j] \right)^2,
$$

(10)

where $W_*$ is obtained from either the SGE or the CSGE of $A_1$, $F_*(x)$ is one of the parameter matrices from Section 4.3.2 (for time step $t = 1$), $\Delta_1 = A_2 - A_1$ contains the new links, the notation $[\cdot]_{i,j}$ denotes the $i,j$ entry of the argument, and $S$ is a sample set. In the ideal case one should set $S = N_1 \cup Z_1$, i.e. the model parameters $x$ should be learned over the set of new links $N_1$ and the set $Z_1$. With this approach, the edges that already exist in $E_1$, i.e., the non-zeros in $A_1$, will not contribute to the learning process. The reason behind this choice of the sample set is that we only want to target links that will form during the next time interval and we do not want to target links that already exist. Unfortunately, for the social networks we consider, the choice $S = N_1 \cup Z_1$ yields a sample set of the order $|V|^2$ and practically impossible to work with. To make the problem manageable, we choose $S$ to contain a fraction of $N_1$ and a fraction of $Z_1$. In addition, $|S|$ should be large enough to capture the essence of the model, but also have manageable size. In our experiments we have $|V| \approx 2 \times 10^6$ and we choose $|S| \approx 5 \times 10^5$ (see Section 5).

Using the SGE $A_1 \approx U_1 A_2 U_1^T$, the CSGE $A_1 \approx V_1 S_1 V_1^T$, and the models from Section 4.3.2, we obtain the following four objectives:

$$
\min_{x \in \mathbb{R}^n} \sum_{(i,j) \in S} \left( [U_1 \text{diag}(x_1, \cdots, x_r) U_1^T]_{i,j} - [\Delta_1]_{i,j} \right)^2,
$$

(11)

$$
\min_{x \in \mathbb{R}^n} \sum_{(i,j) \in S} \left( [V_1 Q_1 \text{diag}(x_1, \cdots, x_r) Q_1^T V_1^T]_{i,j} - [\Delta_1]_{i,j} \right)^2,
$$

(12)

$$
\min_{x \in \mathbb{R}^n} \sum_{(i,j) \in S} \left( [U_1 \{ \sum_{k=1}^p x_k A_k] U_1^T]_{i,j} - [\Delta_1]_{i,j} \right)^2,
$$

(13)

$$
\min_{x \in \mathbb{R}^n} \sum_{(i,j) \in S} \left( [V_1 Q_1 \{ \sum_{k=1}^p x_k A_k S_1] Q_1^T V_1^T]_{i,j} - [\Delta_1]_{i,j} \right)^2.
$$

(14)

In the objectives (12) and (14) we use the full eigendecomposition $S_1 = Q_1 A S_1 Q_1^T$. Each objective constitutes a least squares problem and can be transformed to a more familiar form

$$
\min_x \|M_* x - b\|_2^2,
$$

(15)

where $M_*$ is a different matrix for each objective based on the involved factors, e.g., $U$, $V$, or $Q$, while $b$ is based on $\Delta_1$. But care must be taken in order to solve the least squares problems efficiently. We will consider each case separately. First we make a useful observation. The minimization in the objectives (11)–(14) is taken over a sample set $S$. Then it easy to see that the $(i,j)$’th entry of a matrix product $UDV^T$ for a diagonal matrix $D = \text{diag}(x_1, \cdots, x_k)$ is

$$
[UDV^T]_{i,j} = u_i^T D v_j = (u_i \circ v_j)^T x,
$$

(16)
where \( u_j^T = U(i, :) \) is the \( i \)’th row of \( U \), \( v_j^T = V(j, :) \) is the \( j \)’th row of \( V \), the symbol \( \circ \) denotes the Hadamard (or element-wise) product, and \( x = [x_1, \ldots, x_\ell]^T \). In the following we assume \( S = \{(i_1, j_1), (i_2, j_2), \ldots, (i_{|S|}, j_{|S|})\} \) and set \( r = [i_1, \ldots, i_{|S|}] \) and \( c = [j_1, \ldots, j_{|S|}] \).

1. **Spectral learning with SGE.** Using (16) it follows that \( M \) in the spectral learning model has the form

\[
M_{\text{sl-sge}} = U_c \circ U_r,
\]

where \( U_c = U_1(c, :) \) and \( U_r = U_1(r, :) \) are \( |S| \times k \) matrices. The associated least squares problem should preferably be solved using the QR-factorization of \( U_c \circ U_r \).

2. **Spectral learning with CSGE.** Let now \( A_1 \approx V_1^T D_1 V_1^T \) be the CSGE, and in addition introduce the full eigendecomposition \( D_1 = Q A Q^T \). As before \( F(x) = \text{diag}(x) \) then

\[
M_{\text{cal}} = (V_c Q) \circ (V_l Q).
\]

An important difference with the regular spectral learning is the size of the least squares equations. It is no longer feasible to compute a QR factorization since \( V_c \) is an \( |S| \times cn \) matrix (compared to the \( m \times n \) matrix \( U_c \)). \( V_c \) has a block diagonal structure, therefore \( V_c \) will have a structure as well. It is also infeasible to compute \( V_c Q \) since the result will be a dense matrix. The least squares problem is solved by solving the corresponding normal equations

\[
\min \| M_{\text{cal}}^T M_{\text{cal}} x - M_{\text{cal}}^T b_k \|.
\]

The product

\[
M_{\text{cal}}^T M_{\text{cal}} = ((V_c Q) \circ (V_l Q))^T ((V_c Q) \circ (V_l Q))
\]

has dimensions \( cn \times cn \) and computed elementwise by only using \( V_1 \) and \( Q \), without forming any intermediate factors.

3. **Polynomial learning.** Polynomial learning is very closely related to the Katz measure. Consider the truncated approximation of Katz measure (1). Replacing the single parameter \( \beta \) with a different one \( x_i \) for each power term we obtain a polynomial of degree \( p \). Given the eigen approximation \( A_1 \approx U_1 \Lambda_1 U_1^T \) and \( F(x) = \sum_{i=1}^{p} x_i \Lambda_i^T \) the coefficient matrix becomes

\[
M_{\text{pl}} = (U_c \circ U_r) [l_1 \cdots l_p] \triangleq (U_c \circ U_r) L,
\]

where \( U_c \) and \( U_r \) are as before, and \( l_i = \text{diag}(\Lambda_i^T) \), \( i = 1, \ldots, p \). The structure of \( M_{\text{pl}} \) follows immediately by observing that

\[
[U \left( \sum_{k=1}^{p} x_k \Lambda_k^T \right) V]^T_{i,j} = u_i^T \left( \sum_{k=1}^{p} x_k \Lambda_k^T \right) v_j = (u_i \circ v_j)^T [l_1 \cdots l_p] [x_1 \cdots x_p]^T = (u_i \circ v_j)^T L x.
\]

Note that \( L \) is a \( k \times p \) matrix and \( x = [x_1 \cdots x_p]^T \) the vector of model parameters. It follows that \( M_{\text{pl}} \) has dimensions \( |S| \times p \) where \( p \) usually quite small (at most in the order of tens), and may be explicitly formed prior solving the least squares equation (15).

4. **Clustered polynomial learning.** Using the clustered spectral graph embedding (7) \( A \approx V D V^T \), the eigendecomposition \( D = Q A Q^T \), and a similar analysis as in (17), we obtain the coefficient matrix

\[
M_{\text{cpl}} = ((V_c Q) \circ (V_l Q)) [\tilde{l}_1 \cdots \tilde{l}_p] \triangleq ((V_c Q) \circ (V_l Q)) \tilde{L}
\]

where again \( \tilde{l}_i = \text{diag}(\tilde{\Lambda}_i^T) \) but now yielding \( \tilde{L} \) with dimensions \( cn \times p \). As in the previous case, \( M_{\text{cpl}} \) has dimensions \( |S| \times p \) and may be formed explicitly. Note, however, that the computation of both \( M_{\text{pl}} \) and \( M_{\text{cpl}} \) only involves the matrices \( U, L \) and \( V, Q, \tilde{L} \), respectively.
4.3.4 Link prediction

The different models are validated by predicting the new links in \( A_3 \). The prediction scores are based on

\[
P_\star = \bar{W}_\star F_\star \bar{W}_\star^T,
\]

where \( \bar{W}_\star \) is obtained from a SGE or CSGE based on \( A_2 \) (instead of \( A_1 \) used in the model learning step). Specifically, let the SGE of \( A_2 \) be \( U_2 A_2 U_2^T \), then the prediction scores for the spectral learning with SGE may be written as

\[
P_{\text{sl-sge}} = U_2 F_{\text{sl-sge}}(x) U_2^T = U_2 \text{diag}(x_1, \cdots, x_r) U_2^T,
\]

where the parameter vector \( x = [x_1, \cdots, x_r]^T \) is learned by solving (10). Similarly, let the CSGE of \( A_2 \) be \( V_2 S_2 V_2^T \), then the prediction scores for the spectral learning CSGE are given by

\[
P_{\text{sl-csge}} = V_2 F_{\text{sl-csge}}(x) V_2^T = V_2 Q_2 \text{diag}(x_1, \cdots, x_r) Q_2^T V_2^T,
\]

where \( Q_2 \) is obtained from the eigendecomposition \( S_2 = Q_2 A_2 Q_2^T \), and the parameter vector \( x = [x_1, \cdots, x_r]^T \) is learned by solving a corresponding least squares problem.

4.3.5 Alignment

Note that when training a model we use the eigenspace \( U_1 \) or \( V_1 \) which are both obtained from \( A_1 \). After fitting the model parameters to the least squares objective, we make predictions using \( U_2 \) or \( V_2 \). \( U_1 \) is different from \( U_2 \), and similarly \( V_1 \) is different from \( V_2 \). The eigenspaces \( U_1, U_2 \), and the clustered eigenspaces \( V_1, V_2 \) are close to each other since the number of edges in \( \Delta_1 \) is small compared to the number of edges in \( A_1 \) and \( A_2 = A_1 + \Delta_1 \). But the difference in the order of the eigenvectors may be significant. To address this issue, we re-align the eigenvectors in \( U_2 \) to better fit the eigenvectors in \( U_1 \). One method is to insert the product \( U_2^T U_1 \):

\[
P_{\text{sl-sge}} = U_2 (U_2^T U_1) \text{diag}(x_1, \cdots, x_r) (U_2^T U_1)^T U_2^T,
\]

and similarly to the other learning methods. Our experience suggests that the alignment step gives substantial improvement over unaligned case.

5 Evaluation

In this section we present experimental results that evaluate accuracy, scalability, and flexibility of CSGE in the context of proximity approximation, missing link inference, and link prediction on social networks.

5.1 Dataset description

In our experiments we use three large real-world online social networks with millions of nodes: Flickr [16], LiveJournal [34] and MySpace [41]. LiveJournal and MySpace datasets are obtained from [51], and the Flickr dataset is collected by [38]. From the dataset, we see that the majority (80%) of links in \( P_1 \) are between user pairs who are two-hops away, whereas the fraction of new links between users who are 4 or more hops away is very small. All users in the datasets are connected to at least one other user and for simplicity, we do not consider rare occasions of link deletions. Links in MySpace are undirected because it requires mutual agreement in order for users to become friends. Although Flickr and LiveJournal allow directed “follower/followee” relationships, the majority is symmetric. Therefore, to simplify our evaluation, we make Flickr and LiveJournal undirected.

Flickr: Flickr [16] is a photo-sharing website, where users can connect to each other by indicating a relationship. This dataset was gathered by a breadth-first search on the graph starting from a few seed nodes. To allow most nodes to be discovered, we use the first few months as a bootstrap period, and create snapshots when most nodes have been discovered and link growth has stabilized. Because of this crawling methodology, we observe that even though the snapshot dates are just ten days apart, there is 2% growth in the number of links.
Table 2: Summary of the online social network datasets.

<table>
<thead>
<tr>
<th>Network</th>
<th>Date</th>
<th># of nodes</th>
<th># of links</th>
<th># of added links (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4/14/2007</td>
<td>1,990,149</td>
<td>41,302,536</td>
<td>–</td>
</tr>
<tr>
<td>Flickr</td>
<td>4/25/2007</td>
<td>1,990,149</td>
<td>42,056,754</td>
<td>754,218 (1.8%)</td>
</tr>
<tr>
<td></td>
<td>5/6/2007</td>
<td>1,990,149</td>
<td>42,879,714</td>
<td>822,960 (1.9%)</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>2/16/2009</td>
<td>1,770,961</td>
<td>83,663,478</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>3/4/2009</td>
<td>1,770,961</td>
<td>84,413,542</td>
<td>750,064 (0.8%)</td>
</tr>
<tr>
<td></td>
<td>4/03/2009</td>
<td>1,770,961</td>
<td>85,713,766</td>
<td>1,300,224 (1.5%)</td>
</tr>
<tr>
<td>MySpace</td>
<td>12/11/2008</td>
<td>2,137,264</td>
<td>90,333,122</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>1/11/2009</td>
<td>2,137,264</td>
<td>90,979,264</td>
<td>646,142 (0.7%)</td>
</tr>
<tr>
<td></td>
<td>2/14/2009</td>
<td>2,137,264</td>
<td>91,648,716</td>
<td>669,452 (0.7%)</td>
</tr>
</tbody>
</table>

**LiveJournal**: LiveJournal [34] is a blogging site, where members can become “fans” of other members. We obtained the LiveJournal dataset from [51]. The dataset was collected by listening to the RSS server, which sends out recent update information. The statistics suggest that LiveJournal users are more active in forming links than users in other networks.

**MySpace**: MySpace [41] is a social networking site for people to interact with their acquaintances by posting on each other’s personal pages. We obtained this dataset from [51]. The dataset was created by collecting information of the first 10 million user IDs. MySpace assigns user IDs chronologically. Since the first few million IDs we crawled are also the oldest MySpace users, they have already formed most of the links and are relatively dormant in forming new ones (i.e., the fraction of new links is smaller than those of other networks).

Here, we provide a detailed description of our dataset. Table 2 summarizes some characteristics of three snapshots $A_1$, $A_2$, and $A_3$ for Flickr, LiveJournal, and MySpace dataset we consider in Section 5. In Figure 3 we plot the shortest hop distance of user pairs in $A_1$ who become connected in $A_2$ (i.e., the positive set $P_1$). The figure shows that for all three datasets, only about 80% of users are two hop apart. This suggests that considering only the users with two hop distances can be a cost-effective alternative to the consideration of all user pairs.

![Figure 3: Shortest path distance of positive sets $P_1$.](image)
$c = \sum_{i} m_i / c$

Table 3: Clustering results with GRACLUS; $c$ is the number of clusters and $\mu$ is the average cluster size.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Prox. Embedding</th>
<th>SGE</th>
<th>CSGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGE (all $A_{ii}$)</td>
<td>$S$ Total</td>
<td>$S$ Total</td>
</tr>
<tr>
<td></td>
<td>$S$ Total</td>
<td>$S$ Total</td>
<td>$S$ Total</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$\mu$</td>
<td>Links in $A_{ii}$</td>
</tr>
<tr>
<td>Flickr</td>
<td>18</td>
<td>110,563</td>
<td>71.8%</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>17</td>
<td>106,241</td>
<td>72.5%</td>
</tr>
<tr>
<td>MySpace</td>
<td>17</td>
<td>125,721</td>
<td>51.9%</td>
</tr>
</tbody>
</table>

Table 4: Comparison of preparation times ($c = 20$ and $r = r_i = 100$). All timings are in minutes.

5.2 Scalability

In this section, we compare various aspects of the computational efficiency of CSGE, SGE, and proximity embedding introduced in [51]. All benchmarking were done using an Intel Xeon™ E5440 machine with 32GB memory, running Ubuntu Linux Kernel v2.6.

5.2.1 Graph clustering

In our experiments we use both GRACLUS [12] and METIS [1] to partition the social networks. The two software packages produce different clusterings as they minimize different objective functions: METIS attempts to produce clusters with equal size, regardless of the inherent clustering structure of the network, whereas GRACLUS produces more balanced clusters. Both software produce good quality partitioning in a relatively short period of time. All experiments are conducted on the largest connected component of each graph, which results in only a very small fraction of users (5.1%) and links (0.5%) being discarded. Table 3 gives an example of clustering results on each data set. In this particular case the clustering was performed using GRACLUS in a recursive way until all cluster sizes were smaller than $1/10$ of the original network size. We observe that more than 70% of the links in Flickr and LiveJournal are within the same clusters, while only 51.9% of the links in MySpace are within the same clusters.

5.2.2 Timing benchmarks

Approximation of proximity measures involves two stages: (1) preparation of the embedded matrix and (2) querying proximity of user pairs from the embedded matrix. We evaluate the timing performance of CSGE by comparing it against other algorithms for each of these stages.

In the first stage of preparing an embedded matrix, we compare CSGE against SGE and Proximity Embedding from [51]. In the case of CSGE, the preparation involves three sub-stages: (i) clustering; (ii) SGE of each cluster; and (iii) computation of the embedded adjacency matrix $S$. We compare CSGE and SGE for embedding both the adjacency matrix $A$ and the normalized adjacency matrix $T = D^{-1/2} AD^{-1/2}$

Table 5: Comparison of query times ($r = r_i = 100$, 0.6 million samples).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Direct Method</th>
<th>Proximity Embedding</th>
<th>SGE</th>
<th>CSGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>15.9ms</td>
<td>8040ms</td>
<td>0.051ms</td>
<td>0.042ms</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>23.5ms</td>
<td>14790ms</td>
<td>0.045ms</td>
<td>0.038ms</td>
</tr>
<tr>
<td>MySpace</td>
<td>24.5ms</td>
<td>16655ms</td>
<td>0.076ms</td>
<td>0.040ms</td>
</tr>
</tbody>
</table>
in each of these sub-stages. For Proximity Embedding, which does not directly approximate \( A \) or \( T \), we instead generate models to compute approximations of \( P_{rz} \) and \( P_{rpr} \) with relatively few (1,600) “landmark” nodes so that they are comparable to the embeddings of \( A \) and \( T \), respectively.

Table 4 summarizes the preparation timing benchmarks for the three approximation algorithms for all three data sets. The number of clusters is set to be \( c = 20 \) and the rank in SGE and CSGE is set to be \( r = r_i = 100 \). We see that CSGE outperforms the other two algorithms in most cases; in the approximation of \( A \), CSGE is up to 30% faster than the other algorithms. In the approximation of \( T \), the timing difference between CSGE and SGE becomes even higher, resulting in over an order of magnitude difference. The convergence of iterative spectral methods is influenced by the size of the “gap” in the eigenvalues [28]. Because the eigenvalues of \( T \) are normalized to be between 1 and -1 with very small gaps between them, the computation times for \( T \) are expected to become much longer than that of \( A \); therefore, the difference in timing performance becomes more dramatic. In an extreme case of Flickr dataset where all 100 dominant eigenvalues of \( T \) (and \( T_{ii} \)) are very close to 1, the computation of \( T \) by CSGE is 20% slower than the computation of rooted PageRank by Proximity Embedding due to the aforementioned reasons.

Among the three sub-stages of CSGE, the majority of time is spent in SGE for each of the \( A_{ii} \) and \( T_{ii} \). Figure 4 further analyzes the SGE time with varying numbers of clusters and sizes of embedding dimensions within each cluster. The timing results for \( c = 1 \) are simply SGE on the entire \( A \) (because \( A \) is a single connected component). The timings for \( c > 1 \) are the sum of SGE times for all \( A_{ii} \). We observe from the graph that, for any given \( r \), the aggregate times of cluster-wise approximations are significantly less than the SGE time without clustering. It is also evident that computational time generally decreases with increasing number of clusters. Furthermore, from these results, we can estimate the potential amount of time improvement when we parallelize the per-cluster SGE computation (as discussed in Sec. 3.3).

In the second stage of querying proximity of user pairs, we compare the average query times of 600,000 randomly sampled node pairs for the three embedding algorithms (i.e., CSGE, SGE, and Proximity Embedding) as well as directly calculated common neighbor and Katz scores. We consider the direct calculation in the query stage but not in the preparation stage because the storage of exact proximity measures for all \( m^2 \) node pairs in preparation stage is prohibitive. From Table 5, we observe that all three embeddings are orders of magnitude faster than the direct calculations. While all three embedding algorithms exhibit a millisecond level of fast query time, CSGE is slightly slower than the other two. This is because CSGE is dependent on the size of its dense core matrix \( S \). While the query time of CSGE is already several folds faster than that of Proximity Embedding, in a time-sensitive online application, the query time can be further improved by either using a smaller \( S \) or applying low-rank approximation on \( S \) (as described in further embedding paragraph of Section 3.3).

5.2.3 Scalability of graph embedding

While CSGE has more efficient precomputation, its major advantage is lower memory usage so that it can achieve higher accuracy for a fixed memory cost. In Section 3.2.1, we discussed the memory usage of CSGE. To show the scalability, both in computational time and memory efficiency, we present some measurements on even bigger (artificial) networks. Let \( A_{FL}, A_{LJ}, \) and \( A_{MS} \) be the adjacency matrices for Flickr, LiveJournal and MySpace, respectively. Then we form

\[
B = \text{diag}(A_{FL}, A_{LJ}, A_{MS}) + A_{Off}
\]

where \( A_{Off} \) contains a small fraction of links in the off-diagonal part so that \( B \) becomes connected. We also form \( C \) in a similar way as follows:

\[
C = \text{diag}(B, B) + B_{Off}.
\]

One may consider that the constructed datasets correspond to some real-world social networks. The approximate size, number of links, and number of clusters are presented in the upper half of Table 6.

The lower part of Table 6 shows the memory required to store \( V = \text{diag}(V_1, \cdots, V_c) \) and the embedded adjacency matrix \( S \). Corresponding computational times are also given. We fix \( r_i = 100 \) in all cases and examine the behavior by increasing the number of clusters \( c \) rather than the size of individual clusters. The
<table>
<thead>
<tr>
<th></th>
<th>$A_{PL}$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>2,000,000</td>
<td>6,000,000</td>
<td>12,000,000</td>
</tr>
<tr>
<td>Numbers of links</td>
<td>40,000,000</td>
<td>239,000,000</td>
<td>598,000,000</td>
</tr>
<tr>
<td>$c$</td>
<td>18</td>
<td>52</td>
<td>104</td>
</tr>
<tr>
<td>Timing</td>
<td>22.6 min</td>
<td>116.3 min</td>
<td>391.2 min</td>
</tr>
<tr>
<td>Memory usage</td>
<td>74 MB</td>
<td>546 MB</td>
<td>1,197 MB</td>
</tr>
</tbody>
</table>

Table 6: Computational time and memory usage for CSGE. Network size $m$ and number of links are approximate.

Figure 4: Comparison of eigendecomposition times (on LiveJournal).

memory and time required to create the embedded adjacency matrix $S$ not only grows quadratically with the number of clusters $c$, but also with the rank $r_i$. Despite the fact that the size of $S$ is inherently quadratic in $c$, CSGE is able to handle the largest network with 12 million users and 598 million links - with only 1.2 GB of memory. On the other hand, regular SGE is unable to load $U$ for $B$ network, confirming that $S$ is indeed not a limiting factor of memory usage in CSGE, and that it has higher spatial scalability than SGE.

5.3 Proximity measure estimation

Next, we compare the accuracy of Katz, rooted Page-Rank, and escape probability when they are combined with SGE and CSGE. For the sake of brevity, we present results only for LiveJournal.

5.3.1 Evaluation methodology

Since it is expensive to compute and store the actual proximity measures for all $m^2$ user pairs, we instead evaluate the two methods using a sampled user pairs $S$, which consists of 100 columns randomly chosen from the proximity matrix. This gives roughly 180,000,000 sample points. For each pair $(i,j) \in S$ we let $p_{ij}$ denote the ‘true’ value of a proximity measure and $\hat{p}_{ij}$ to be an ‘estimate’. For example, we may have $p_{ij} = P_{kz}[i,j]$ while $\hat{p}_{ij} = P_{kz,sge}[i,j]$. The true proximity measures were computed for comparison purposes using the methodology outlined in [51]. Regarding the accuracy measure, we use the normalized absolute error $e_{ij} = |p_{ij} - \hat{p}_{ij}|/\mu$, where $\mu = \sum_{(i,j)\in S} p_{ij}/|S|$. We plot the cumulative distribution function (CDF) to compare the error.

In all Katz measure computations, we use $\beta = 0.0005$ and $k_{max} = 6$. For computing the rooted PageRank and the escape probability, we use $\alpha = 0.15$ and $k_{max} = 20$. Recall that the different proximity measures
are described in Table 1. The embedding subspace dimensions in SGE and within each cluster in CSGE are set to \( r = 100 \). Clustering of the datasets is done on \( A \) while the graph embedding is based on \( A \) for Katz measure and \( T = D^{−1/2}AD^{−1/2} \) for rooted PageRank and escape probability measures.

### 5.3.2 Accuracy evaluation

We consider approximating a low-rank matrix. Figure 5 (a) plots the CDF of the normalized absolute errors in approximating Katz measure with \( P_{kz, sge} \) and \( P_{kz, csg} \). We make two observations: (1) for most samples, the error is small: with 85% of the node pairs having an error of 0.2 or less; and (2) the error for CSGE is lower than SGE with the gap in error as little as 2%. A likely reason for SGE and CSGE yielding similar performance is that \( P_{kz} \) has a low intrinsic dimensionality and that both SGE and CSGE have reached a point where \( r = 100 \) is enough for the low-rank approximation of Katz.

We consider now approximation of a non low-rank matrix. In Figure 5 (b) we present the CDF of normalized absolute errors in approximating the rooted PageRank measure with \( P_{rpr, sge} \) and \( P_{rpr, csg} \). We observe that clustering gives a considerable improvement in the accuracy of the rooted PageRank measure. For over 95% of the samples, the normalized absolute error is less than 0.01. On the other hand, SGE exhibits a relatively higher error: above 0.6 error for 95% of the samples. It can be verified that the normalized adjacency matrix \( T \), for which we compute SGE and CSGE, has a much higher intrinsic dimensionality than \( A \). The improved accuracy in \( P_{rpr, csg} \) may be explained by the fact that 100 dimensional embeddings on each cluster (CSGE) captures a much larger fraction of variance compared to an \( r = 100 \) embedding on the entire matrix (SGE). Figure 5 (c) shows the CDFs of normalized absolute error for escape probability measure \( P_{ep, sge} \) and \( P_{ep, csg} \). Again, using clustering significantly improves the accuracy. This improvement is to be expected because the escape probability is based on the rooted PageRank measure. While CSGE provides a normalized absolute error less than 0.1 for more than 95% of the samples, SGE exhibits a much higher error of over 20 for the same 95% of the samples.

As a summary, our proximity estimation evaluation shows that CSGE is not only effective in approximating proximity measures on \( A \), but also performs well on matrices with high intrinsic dimension such as the normalized adjacency matrix \( T \). Compared with SGE, CSGE can accurately approximate the rooted PageRank and escape probability despite the general difficulty in approximating these metrics.

### 5.4 Missing link inference

An important problem in social network applications is missing link inference. We evaluate the accuracy of inferring different proximity measures with and without clustering.
5.4.1 Evaluation methodology: metrics and experimental setup

The accuracy of link prediction is quantified by computing false positive rates (FPR = \( \frac{\# \text{of incorrectly predicted friend links}}{\# \text{of non-friend links}} \)) and false negative rates (FNR = \( \frac{\# \text{of missed friend links}}{\# \text{of new-friend links}} \)) of all user pairs in a sample set. Note that the denominator of the FPR is the number of user pairs who are not friends. This number is usually very large, e.g., in MySpace, we have \( 2 \times 10^6 \times 2 \times 10^6 - 90 \times 10^6 \approx 10^{12} \). Since we are more interested in picking up small number of correct friendships (as opposed to finding as many new friendship links as possible), we present trade-off curves with an emphasis to small FPR area by displaying the x-axis in log-scale.

From the first snapshots of all three datasets, we randomly mask half of the links as missing \( M \). The adjacency matrix with the remaining links is considered as an incomplete adjacency matrix with observed links \( A_M \). The trade-off curve between FPR and FNR is plotted for all user pairs given the non-zero of \( A_M \).

5.4.2 Accuracy evaluation

In Figure 6, we present the performance of \( P_{csnge} \), \( P_{cr-csge} \), \( P_{kz-csge} \), and \( P_{kz-csge} \). We observe that using CSGE in both proximity measures consistently outperforms SGE in all three datasets. Comparing across different measures, we observe that Katz measure with CSGE generally performs the best. For instance, in LiveJournal, for a given FPR, \( P_{kz-csge} \) yields 10% or less FNR than \( P_{kz-sge} \).

5.5 Link prediction evaluation

In this section, we compare the performance of supervised clustered and non-clustered spectral link prediction method against unsupervised proximity measures. Through our performance evaluation, we verify that learning model parameters in a supervised way is indeed helpful in improving accuracy within all datasets.

5.5.1 Evaluation methodology

Table 2 displays information related to the three snapshots \( A_1 \), \( A_2 \), and \( A_3 \) of each dataset. To learn the model parameters in the two supervised models, we use a graph embedding of \( A_1 \) and minimize Eq. 10 by explicitly targeting newly formed links in \( A_2 \). In the next step, we use an embedding of \( A_2 \) with the learned model parameters to predict new links in \( A_3 \).

Because of the size of our datasets, we randomly select a fraction of the positive and negative samples. Specifically, for training, we select 100,000 user pairs from the positive links \( P_1 \), and 500,000 from the negative links \( N_1 \). For testing, we pick a different sample set of the same size as before but now from \( P_2 \) and \( N_2 \).

In a second experiment, we learn the model parameters based on a sample set of user pairs that are connected by two hops. This practical scenario focuses on link prediction for user pairs who are already close in the network (likely to be friends but not friends yet) and thus require proximity algorithms to consider.
only a small set of user pairs. These are user pairs that will form a triangle with a common friend, if they become friends. Figure 3 in appendix provides proportion of users with two hop distances for each dataset.

The accuracy of link prediction is quantified using the FPR and FNR introduced in Section 5.4. In the context of link prediction, the “true” and “estimated” links refer to the links in $P_2$. As in missing link inference, the raw count of non-friend pairs in FPR is extremely large, and thus we present the performance measures with x-axis in log-scale.

We have conducted an extensive set of experiments, with numerous methods. In the interest of brevity, we present link prediction results in Figure 7 and 8 based on three link models: $P_{cn}$, $P_{kz-sge}$ and the supervised $P_{sl-csge}$.

5.5.2 Accuracy evaluation

Figure 7 presents the link prediction accuracy of $P_{cn}$, $P_{kz-sge}$ and $P_{sl-csge}$. We see that, the spectral learning with CSGE performs the best in all datasets, followed by the Katz measure with SGE. For example, at an FPR of 0.001, spectral learning with CSGE reduces FNR by more than 10% in both Flickr and LiveJournal. In MySpace, the performance of clustered spectral learning is still better than the other two measures albeit by a smaller margin.

In Figure 8, we evaluate link predictors for user pairs who are only two hops apart. Our spectral learning model again outperforms other measures by up to 20% in the Flickr dataset, and 10% in the LiveJournal dataset. For MySpace, the performance of Katz with SGE and spectral learning with CSGE is better than common neighbor by about 4%.

Note that there is no significant difference among predictors in MySpace. We speculate that this is an artifact of the data collection technique. Since this data set contains first 10 million users [51], these users have been in the network the longest time, and seem to already have a large number of friends, and thus are less active in creating new relationships. An indication of the claim is in Table 2 exhibiting the relatively small rate of link increase in MySpace compared to Flickr and LiveJournal. Another possible reason is the large number of inter-cluster links in MySpace, as shown in Table 3. Taking into account that 48% of the links are outside the clusters, the benefit of the clustering approach may have diminished, as almost half the links in the network are not used when computing the cluster-wise embeddings.

This section shows that our spectral learning algorithm combined with CSGE is flexible and consistently out-performs the existing schemes across different datasets and proximity measures.

6 Related Work

In this section, we first survey related work on social network analysis in general, focusing primarily on research done by the networking and systems research community. We then survey related work in the context of proximity estimation, link prediction, and missing link inference.
6.1 Social network analysis

Traditionally, studies on social networks often focus on relatively small social networks (e.g., [33] examine co-authorship networks with about 5000 nodes). The recent explosion of online social networks, however, has given rise large-scale social networks with billions of links. A number of measurement studies characterize the topological structure, information propagation and user behavior of online social networks (e.g., [39, 38, 2, 54, 8, 9]). There have also been a number of efforts on leveraging social networks to defend against Sybil attacks [59, 55], fight spams [17], build socially aware systems [44], and improve Internet search [37]. In [49], we explore the use of graph clustering with a few different within clusters dimensionality reduction schemes. In this paper, we develop the CSGE and demonstrate its effectiveness on large scale social network analysis tasks, and illustrate its ability to cope with temporal dynamics present in real world applications.

6.2 Spectral embedding

Spectral decomposition or spectral embedding provided by spectral theorem is central to our approach. Eigendecomposition seeks to decompose a square matrix into product of eigenvector matrices and a diagonal eigenvalue matrix. There is a large body of literature dedicated to finding computationally efficient ways for eigendecomposition [22]. Golub and Van Loan [18] provide a numerically stable and fast method for eigendecomposition of matrices. A variety of software packages implements optimized eigendecomposition, such as PROPACK [27] and ARPACK [28] for MATLAB.

6.3 Clustering

While clustering in itself is not addressed in this paper, fast clustering is crucial to the methods that we propose. Clustering algorithms can be categorized by their objective function for graph cuts, like \textit{minimum cut} [57], \textit{ratio cut} [19], \textit{normalized cut}[50]. These problems are shown to be NP hard [56]. Relaxing the graph partitioning problem leads to spectral clustering objectives that, involves computation of eigenvectors of the normalized or unnormalized graph laplacians; A good tutorial is provided here [35]. This research has enabled the development of state of the art software, like GRACLUS [25] and METIS [1], that approximately solve the graph partitioning problem without explicitly computing eigenvectors [12, 1]. Although our approach is valid for any clustering, most computational benefits are obtained when cluster sizes are approximately the same, while capturing the largest number of edges within the clusters. With respect to this, the \textit{normalized cut} objective [50], which seeks to minimize dis-associations between clusters and maximize associations within a cluster, is well suited for the clustering problem.

6.4 Proximity estimation

Given their importance for social network applications, proximity measures have received considerable research attention (e.g., [21, 33, 40, 47, 52, 48, 23]). A number of proximity measures have been proposed,
such as common neighbors, the Katz measure [21], rooted PageRank [33], and escape probability [52]. These proximity measures have been successfully applied to a wide range of social network applications (e.g., fraud detection [10], viral marketing [20], and link prediction [33]). Despite their effectiveness, many existing proximity measures are computationally prohibitive for large social networks [48, 52]. Recently, Song et al. [51] proposed scalable techniques for approximating proximity measures in online social networks based on the idea of low-rank approximation. Experiments show that our new technique is not only much more scalable, but also achieves dramatic improvements in accuracy when the proximity measures of interest have high intrinsic dimensionality and thus do not have good low-rank approximations.

6.5 Link prediction

The problem of link prediction in social networks was first introduced in [32] and a number of proximity measures have been considered for link prediction in [32, 33]. A decision tree based link predictor that combines multiple proximity measures has been studied in [51]. There has also been some recent work on using supervised learning methods for link prediction in diverse networks such as hyperlink and citation graphs [26]. Again experiments show that our supervised link prediction technique can achieve considerable improvement in link prediction accuracy.

6.6 Missing link inference

The problem of missing link inference is closely related to link prediction and was also mentioned in [32, 33]. Missing link inference falls into the general realm of compressive sensing, which has broad applicability in computer science (e.g., [13, 46, 7, 60]) . However, we are unaware of any existing compressive sensing techniques that can scale to massive social networks with millions of vertices. As a result, in this paper we focus on simple inference algorithms based on proximity measures, which are much more scalable and can directly benefit from our clustered spectral graph embedding technique.

The extreme class imbalance between negative and positive samples, and the large number of potential links result in low accuracy for existing link prediction algorithms [45, 14]. To solve this, some work focuses on the Anomalous Link Discovery problem, which finds anomalous or interesting links in a graph, which is considered relatively more tractable than the link prediction problem [45], while others [14] come up with learning algorithms based on chance constrained programs, to overcome the skewness.

7 Conclusion

In this paper we propose a novel dimensionality reduction technique called clustered spectral graph embedding (CSGE) that can embed a massive original graph into a much smaller graph. The embedded graph captures the essential clustering and spectral structure of the original graph and allows a wide range of analysis tasks to be performed in an efficient and accurate fashion. State-of-the-art graph embedding techniques, such as spectral graph embedding, often require the underlying graph to have good low-rank approximations, and tend to perform poorly when the low-rank assumption is violated. In contrast, CSGE can easily cope with massive graphs with much higher intrinsic dimensionality and achieve dramatic improvement on both accuracy and efficiency (with respect to computational cost and memory usage).

To evaluate the performance of clustered spectral graph embedding, we explore three important social network analysis tasks (proximity estimation, missing link inference, and link prediction) using three large-scale real-world social network datasets (Flickr, LiveJournal and MySpace; with up to 2 million vertices and 90 million edges). For proximity estimation, we achieve up to an order of magnitude improvement in computation time and memory usage, and up to several orders of magnitude improvement in accuracy when the proximity measures of interest have high intrinsic dimensionality. For missing link inference, CSGE consistently yields better inference on unobserved links across different datasets. For link prediction, our novel supervised learning automatically learns parameter configurations optimal to target networks,
achieving the best accuracy among proximity measures being compared. These results clearly demonstrate the effectiveness and the potential values of our approach.

References


