Journal of Complex Networks (2020) **2**, Advance Access Publication on 6 March 2020
doi: 10.1093/comnet/cnaa007

GLEE: Geometric Laplacian Eigenmap Embedding

LEO TORRES[†]
Network Science Institute, Northeastern University, Boston, MA 02115, USA
[†]Corresponding author. Email: leo@leotrs.com

KEVIN S. CHAN
U.S. CCDC Army Research Laboratory, Adelphi, MD 20783, USA

AND

TINA ELIASSI-RAD
Network Science Institute and Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115, USA

Edited by: Ernesto Estrada

[Received on 28 October 2019; editorial decision on 17 January 2020; accepted on 22 January 2020]

Graph embedding seeks to build a low-dimensional representation of a graph G. This low-dimensional representation is then used for various downstream tasks. One popular approach is Laplacian Eigenmaps (LE), which constructs a graph embedding based on the spectral properties of the Laplacian matrix of G. The intuition behind it, and many other embedding techniques, is that the embedding of a graph must respect node similarity: similar nodes must have embeddings that are close to one another. Here, we dispose of this distance-minimization assumption. Instead, we use the Laplacian matrix to find an embedding with geometric properties instead of spectral ones, by leveraging the so-called simplex geometry of G. We introduce a new approach, *Geometric Laplacian Eigenmap Embedding*, and demonstrate that it outperforms various other techniques (including LE) in the tasks of graph reconstruction and link prediction.

Keywords: graph embedding; graph Laplacian; simplex geometry.

1. Introduction

Graphs are ubiquitous in real-world systems from the internet to the world wide web to social media to the human brain. The application of machine learning to graphs is a popular and active research area. One way to apply known machine learning methods to graphs is by transforming the graph into a representation that can be directly fed to a general machine learning pipeline. For this purpose, the task of graph representation learning, or graph embedding, seeks to build a vector representation of a graph by assigning to each node a feature vector that can then be fed into any machine learning algorithm.

Popular graph embedding techniques seek an embedding where the distance between the latent representations of two nodes represents their similarity. For example, [1] calls this the 'community aware' property (nodes in a community are considered similar, and thus their representations must be close to one another), while [2] calls it a 'symmetry' between the node domain and the embedding domain. Others call methods based on this property with various names such as 'positional' embeddings [3] or 'proximity-based' embeddings [4]. Consequently, many of these approaches are formulated in such a way that the distance (in the embedding space) between nodes that are similar (in the original data domain) is small. Here, we present a different approach. Instead of focusing on minimizing the distance between similar nodes, we seek an embedding that preserves the most basic structural property of the graph, namely adjacency; the works [3, 4] call this approach 'structural' node embeddings. Concretely, if the nodes i and j are neighbours in the graph G with n nodes, we seek d-dimensional vectors s_i and s_j such that the adjacency between i and j is encoded in the geometric properties of s_i and s_j, for some $d \ll n$. Examples of geometric properties are the dot product of two vectors (which is a measure of the angle between them), the length (or area or volume) of a line segment (or polygon or polyhedron), the centre of mass or the convex hull of a set of vectors, among others. In Section 3, we propose one such geometric embedding technique, called Geometric Laplacian Eigenmap Embedding (GLEE), that is based on the properties of the Laplacian matrix of G, and we then proceed to compare it to the original formulation of Laplacian Eigenmaps (LE) as well as other popular embedding techniques.

GLEE has deep connections with the so-called simplex geometry of the Laplacian [5, 6]. Fiedler [6] first made this observation, which highlights the bijective correspondence between the Laplacian matrix of an undirected, weighted graph and a geometric object known as a *simplex*. Using this relationship, we find a graph embedding such that the representations s_i, s_j of two non-adjacent nodes i and j are always orthogonal, $s_i \cdot s_j = 0$, thus achieving a geometric encoding of adjacency. Note that this does not satisfy the 'community-aware' property of [1]. For example, the geometric embedding s_i of node i will be orthogonal to each non-neighbouring node, including those in its community. Thus, s_i is not close to other nodes in its community, whether we define *closeness* in terms of Euclidean distance or cosine similarity. However, we show that this embedding—based on the simplex geometry—contains desirable information, and that it outperforms the original, distance-minimizing, formulation of LE on the tasks of graph reconstruction and link prediction in certain cases.

The contributions of this work are as follows:

1. We present a geometric framework for graph embedding that departs from the tradition of looking for representations that minimize the distance between similar nodes by highlighting the intrinsic geometric properties of the Laplacian matrix.
2. The proposed method, GLEE, while closely related to the LE method, outperforms LE in the tasks of link prediction and graph reconstruction. Moreover, a common critique of LE is that it only considers first-order adjacency in the graph. We show that GLEE takes into account higher-order connections (see Section 3.2).
3. The performance of existing graph embedding methods (which minimize distance between similar nodes) suffers when the graph's average clustering coefficient is low. This is not the case for GLEE.

In Section 2, we recall the original formulation of LE, in order to define the GLEE in Section 3 and discuss its geometric properties. We mention related work in Section 4 and present experimental studies of GLEE in Section 5. We finish with concluding remarks in Section 6.

2. Background on LE

Belkin and Niyogi [7, 8] introduced LE as a general-purpose method for embedding and clustering an arbitrary dataset. Given a dataset $\{x_i\}_{i=1}^{n}$, a proximity graph $G = (V, A)$ is constructed with node set $V = \{x_i\}$ and edge weights $\mathbf{A} = (a_{ij})$. The edge weights are built using one of many heuristics that determine which nodes are close to each other and can be binary or real valued. Some examples are k

nearest neighbours, ε-neighbourhoods, heat kernels, etc. To perform the embedding, one considers the Laplacian matrix of G, defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal matrix whose entries are the degrees of each node. One of the defining properties of $\mathbf{L}$ is the value of the quadratic form:

$$y^\top \mathbf{L} y = \frac{1}{2} \sum_{i,j} a_{ij} (y_i - y_j)^2. \tag{2.1}$$

The vector y^* that minimizes the value of (2.1) will be such that the total weighted distance between all pairs of nodes is minimized. Here, y_i can be thought of as the one-dimensional embedding of node i. One can then extend this procedure to arbitrary d-dimensional node embeddings by noting that $tr(\mathbf{Y}^\top \mathbf{L} \mathbf{Y}) = \sum_{i,j} a_{ij} \|y_i - y_j\|^2$, where $\mathbf{Y} \in \mathbb{R}^{n \times d}$ and y_i is the ith row of $\mathbf{Y}$. The objective function in this case is

$$\begin{aligned} \mathbf{Y}^* = \underset{\mathbf{Y} \in \mathbb{R}^{n \times d}}{\arg\min}\, tr(\mathbf{Y}^\top \mathbf{L} \mathbf{Y}) \\ \text{s.t. } \mathbf{Y}^\top \mathbf{D} \mathbf{Y} = \mathbf{I}. \end{aligned} \tag{2.2}$$

Importantly, the quantity $tr(\mathbf{Y}^\top \mathbf{L} \mathbf{Y})$ has a global minimum at $\mathbf{Y} = 0$. Therefore, a restriction is necessary to guarantee a non-trivial solution. Belkin and Niyogi [7, 8] choose $\mathbf{Y}^\top \mathbf{D} \mathbf{Y} = \mathbf{I}$, though others are possible. Applying the method of Lagrange multipliers, one can see that the solution of (2) is achieved at the matrix $\mathbf{Y}^*$ whose rows y_i^* are the solutions to the eigenvalue problem

$$\mathbf{L} y_i^* = \lambda_i \mathbf{D} y_i^*. \tag{2.3}$$

When the graph contains no isolated nodes, y_i^* is then an eigenvector of the matrix $\mathbf{D}^{-1}\mathbf{L}$, also known as the normalized Laplacian matrix. The embedding of a node j is then the vector whose entries are the jth elements of the eigenvectors $y_1^*, y_2^*, ..., y_d^*$.

3. Proposed approach: Geometric LE

We first give our definition and then proceed to discuss both the algebraic and geometric motivations behind it.

Definition 3.1 (GLEE) Given a graph G, consider its Laplacian matrix $\mathbf{L}$. Using singular value decomposition, we may write $\mathbf{L} = \mathbf{S}\mathbf{S}^\top$ for a unique matrix $\mathbf{S}$. Define $\mathbf{S^d}$ as the matrix of the first d columns of $\mathbf{S}$. If i is a node of G, define its d-dimensional *Geometric Laplacian Eigenmap Embedding* (GLEE) as the ith row of $\mathbf{S^d}$, denoted by s_i^d. If the dimension d is unambiguous, we will just write s_i.

Algebraic motivation. In the case of positive semidefinite matrices, such as the Laplacian, the singular values coincide with the eigenvalues. Moreover, it is well known that $\mathbf{S^d}$ is the matrix of rank d that is closest to $\mathbf{L}$ in Frobenius norm, i.e., $\|\mathbf{L} - \mathbf{S^d}(\mathbf{S^d})^\top\|_F \leq \|\mathbf{L} - \mathbf{M}\|_F$ for all matrices $\mathbf{M}$ of rank d. Because of this, we expect $\mathbf{S^d}$ to achieve better performance in the graph reconstruction task than any other d-dimensional embedding (see Section 5.1).

As can be seen from Equation (2.1), the original formulation of LE is due to the fact that the distance between the embeddings of neighbouring nodes is minimized, under the restriction $Y^\top D Y = I$. We can also formulate GLEE in terms of the distance between neighbouring nodes. Perhaps counterintuitively, GLEE solves a distance *maximization* problem, as follows. The proof follows from a routinary application of Lagrange multipliers and is omitted.

THEOREM 3.2 Let Λ be the diagonal matrix whose entries are the eigenvalues of $\mathbf{L}$. Consider the optimization problem

$$\begin{aligned} &\underset{\mathbf{Y}\in\mathbb{R}^{n\times d}}{\arg\max}\, tr(\mathbf{Y}^\top \mathbf{L}\mathbf{Y}) \\ &\text{s.t. } \mathbf{Y}^\top\mathbf{Y} = \Lambda. \end{aligned} \tag{3.1}$$

Its solution is the matrix $\mathbf{S^d}$ whose columns are the eigenvectors corresponding to the largest eigenvalues of $\mathbf{L}$. If $d = n$, then $\mathbf{L} = \mathbf{S^d}\left(\mathbf{S^d}\right)^\top$. □

The importance of Theorem 3.2 is to highlight the fact that distance-minimization may be misleading when it comes to exploiting the properties of the embedding space. Indeed, the original formulation of LE, while well established in Equation 2.2, yields as result the eigenvectors corresponding to the *lowest* eigenvalues of $\mathbf{L}$. However, standard results in linear algebra tell us that the best low rank approximation of $\mathbf{L}$ is given by the eigenvectors corresponding to the *largest* eigenvalues. Therefore, these are the ones used in the definition of GLEE.

Geometric motivation. The geometric reasons underlying Definition 3.1 are perhaps more interesting than the algebraic ones. A recent review paper [5] highlights the work of Fiedler [6], who discovered a bijective correspondence between the Laplacian matrix of a graph and a higher-dimensional geometric object called a *simplex*.

DEFINITION 3.3 Given a set of $k+1$ k-dimensional points $\{p_i\}_{i=0}^{k}$, if they are affinely independent (i.e. if the set of k points $\{p_0 - p_i\}_{i=1}^{k}$ is linearly independent), then their convex hull is called a *simplex*.

A simplex is a high-dimensional polyhedron that is the generalization of a two-dimensional triangle or a three-dimensional tetrahedron. To see the connection between the Laplacian matrix of a graph and simplex geometry, we invoke the following result. The interested reader will find the proof in [5, 6].

THEOREM 3.4 Let $\mathbf{Q}$ be a positive semidefinite $k\times k$ matrix. There exists a $k\times k$ matrix $\mathbf{S}$ such that $\mathbf{Q} = \mathbf{S}\mathbf{S}^\top$. The rows of $\mathbf{S}$ lie at the vertices of a simplex if and only if the rank of $\mathbf{Q}$ is $k-1$. □

COROLLARY 3.5 Let G be a connected graph with n nodes. Its Laplacian matrix $\mathbf{L}$ is positive semidefinite, has rank $n-1$, and has eigendecomposition $\mathbf{L} = \mathbf{P}\Lambda\mathbf{P}^\top$. Write $\mathbf{S} = \mathbf{P}\sqrt{\Lambda}$. Then, $\mathbf{L} = \mathbf{S}\mathbf{S}^\top$ and the rows of $\mathbf{S}$ are the vertices of a $(n-1)$-dimensional simplex called the *simplex of* G. □

Corollary 3.5 is central to the approach in [5], providing a correspondence between graphs and simplices. Corollary 3.5 also shines a new light on GLEE: *the matrix* $\mathbf{S^d}$ *from Definition 3.1 corresponds to the first d dimensions of the simplex of G.* In other words, computing the GLEE embeddings of a graph G is equivalent to computing the simplex of G and projecting it down to d dimensions. We proceed to explore the geometric properties of this simplex that can aid in the interpretation of GLEE embeddings. We can find in [5] the following result.

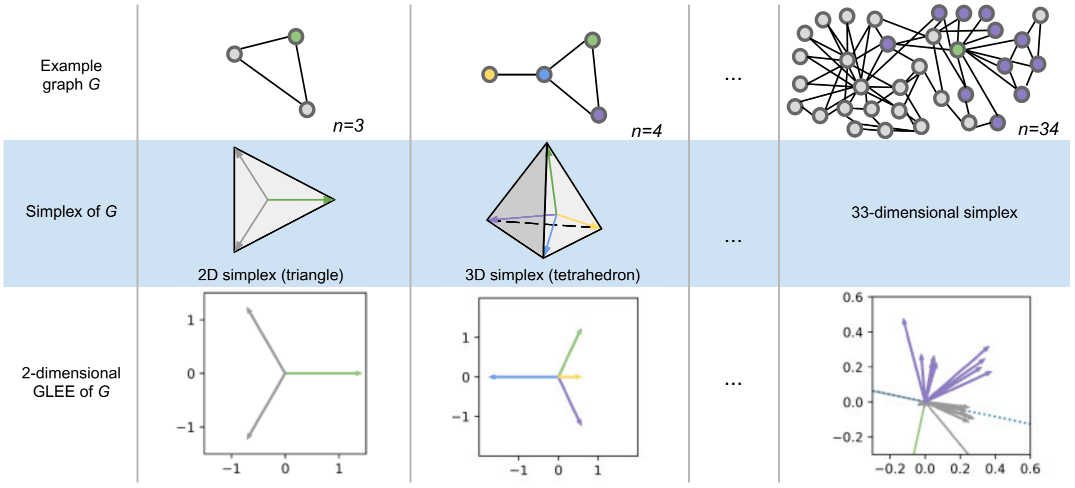

FIG. 1. Simplex geometry and GLEE. Given a graph G with n nodes (top row), there is a $(n-1)$-dimensional simplex that perfectly encodes the structure of G, given by the rows of the matrix S from Corollary 3.5 (middle row). The first d columns of S yield the GLEE of G (bottom row). In each example, embeddings are colour-coded according to the node they represent. For $n = 3$, all nodes in the triangle graph are interchangeable. Accordingly, their embeddings all have the same length and subtend equal angles with each other. For $n = 4$, the green and purple nodes are interchangeable, and thus their embeddings are symmetric. Note that the length of each embedding corresponds to the degree of the corresponding node. For $n = 34$ we show the Karate Club network [9], in which we highlight one node in green and all of its neighbours in purple. In the bottom right panel, the dotted line is orthogonal to the green node's embedding. Note that most of the non-neighbours' embeddings (in gray) are close to orthogonal to the green node's embedding, while all neighbours (in purple) are not.

COROLLARY 3.6 Let s_i be the ith row of $\mathbf{S}$ in Corollary 3.5. s_i is the simplex vertex corresponding to node i, and satisfies $\|s_i\|^2 = \deg(i)$, and $s_i \cdot s_j^\top = -a_{ij}$, where $\deg(i)$ is the degree of i. In particular, s_i is orthogonal to the embedding of any non-neighbouring node j. □

Corollary 3.6 highlights some of the basic geometric properties of the simplex (such as lengths and dot products) that can be interpreted in graph theoretical terms (respectively degrees and adjacency). In Fig. 1, we show examples of these properties. It is worth noting that other common matrix representations of graphs do not present a spectral decomposition that yields a simplex. For example, the adjacency matrix $\mathbf{A}$ is not in general positive semidefinite, and the normalized Laplacian $\mathbf{D^{-1}L}$ (used by LE) is not symmetric. Therefore, Theorem 3.4 does not apply to them. We now proceed to show how to take advantage of the geometry of GLEE embeddings, which can all be thought of as coming from the simplex, in order to perform common graph mining tasks. In the following we focus on unweighted, undirected graphs.

3.1 *Graph reconstruction*

For a graph G with n nodes, consider its d-dimensional GLEE embedding $\mathbf{S^d}$. When $d = n$, in light of Corollary 3.6, the dot product between any two embeddings s_i, s_j can only take the values -1 or 0 and one can reconstruct the graph perfectly from its simplex. However, if $d < n$, the distribution of dot products will take on real values around -1 and 0 with varying amounts of noise; the larger the dimension d, the less noise we find around the two modes. It is important to distinguish which nodes i, j have embeddings

s_i, s_j whose dot product belongs to the mode at 0 or to the mode at -1, for this determines whether or not the nodes are neighbours in the graph. One possibility is to simply 'split the difference' and consider i and j as neighbours whenever $s_i \cdot s_j < -0.5$. More generally, given a graph G and its embedding $\mathbf{S^d}$, define $\mathbf{L}\hat{(}\theta)$ to be the estimated Laplacian matrix using the above heuristic with threshold θ, that is

$$\mathbf{L}_{ij}\hat{(}\theta) = \begin{cases} -1 & s_i \cdot s_j^\top < \theta \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

Then, we seek the value of θ, call it θ_{opt}, that minimizes the loss

$$\theta_{\text{opt}} = \underset{\theta\in[-1,0]}{\arg\min} \|\mathbf{L} - \hat{\mathbf{L}}(\theta)\|_F^2. \tag{3.3}$$

If all we have access to is the embedding, but not the original graph, we cannot optimize Equation (3.3) directly. Thus, we have to estimate θ_{opt} heuristically. As explained above, one simple estimator is the constant $\hat{\theta}_c = -0.5$. We develop two other estimators: $\hat{\theta}_k, \hat{\theta}_g$, obtained by applying Kernel Density Estimation and Gaussian Mixture Models (GMMs), respectively. We do so in Appendix A as their development has little to do with the geometry of GLEE embeddings. Our experiments show that different thresholds θ_c, θ_k, and θ_g produce excellent results on different datasets; see Appendix A for discussion.

3.2 *Link prediction*

Since the objective of GLEE is to directly encode graph structure in a geometric way, rather than solve any one particular task, we are able to use it in two different ways to perform link prediction. These are useful in different kinds of networks.

3.2.1 *Number of common neighbours.*

It is well known that heuristics such as number of CN or Jacard similarity (JS) between neighbourhoods are highly effective for the task of link prediction in networks with a strong tendency for triadic closure [10]. Here, we show that we can use the geometric properties of GLEE in order to approximately compute CN. For the purpose of exposition, we assume $d = n$ unless stated otherwise in this section.

Given an arbitrary subset of nodes V in the graph G, we denote by $|V|$ its number of elements. We further define *the centroid of* V, denoted by C_V, as the centroid of the simplex vertices that correspond to its nodes, i.e., $C_V = \frac{1}{|V|}\sum_{i\in V} s_i$. The following lemma, which can be found in [5], highlights the graph-theoretical interpretation of the geometric object C_V.

LEMMA 3.7 (From [5]) Given a graph G and its GLEE embedding S, consider two disjoint node sets V_1 and V_2. Then, the number of edges with one endpoint in V_1 and one endpoint in V_2, is given by

$$-|V_1||V_2|\ C_{V_1}^\top \cdot C_{V_2}. \tag{3.4}$$

Proof. By linearity of the dot product, we have

$$|V_1||V_2|C_{V_1}^\top \cdot C_{V_2} = \sum_{i\in V_1}\sum_{j\in V_2} s_i \cdot s_j^\top = -\sum_{i\in V_1}\sum_{j\in V_2} a_{ij}. \tag{3.5}$$

The expression on the right is precisely the required quantity. □

Lemma 3.7 says that we can use the dot product between the centroids of two node sets to count the number of edges that are shared by them. Thus, we now reformulate the problem of finding the number of CN between two nodes in terms of centroids of node sets. In the following, we use $N(i)$ to denote the neighbourhood of node i, that is, the set of nodes connected to it.

LEMMA 3.8 Let $i, j \in V$ be non-neighbours. Then, the number of CN of i and j, denoted by $CN(i,j)$, is given by

$$CN(i,j) = -\deg(i)\, C_{N(i)} \cdot s_j^\top = -\deg(j)\, C_{N(j)} \cdot s_i^\top . \tag{3.6}$$

Proof. Apply Lemma 3.7 to the node sets $V_1 = N(i)$ and $V_2 = \{j\}$, or, equivalently, to $V_1 = N(j)$ and $V_2 = \{i\}$. □

Now assume we have the d-dimensional GLEE of G. We approximate $CN(i,j)$ by estimating both $\deg(i)$ and $C_{N(j)}$. First, we know from Corollary 3.6 that $\deg(i) \approx \|s_i^d\|^2$. Second, we define the approximate neighbour set of i as $\hat{N}(i) = \{k : s_k^d \cdot (s_i^d)^\top < \hat{\theta}\}$, where $\hat{\theta}$ is any of the estimators from Section 3.1. We can now write

$$CN(i,j) \approx -\|s_i^d\|^2 C_{\hat{N}(i)} \cdot (s_j^d)^\top . \tag{3.7}$$

The higher the value of this expression, the more confident is our prediction that the link (i,j) exists.

3.2.2 *Number of paths of length 3.* A common critique of the original LE algorithm is that it only takes into account first-order connections, which were considered in Section 3.2.1. Furthermore, authors of [11] point out that the application of link prediction heuristics CN and JS does not have a solid theoretical grounding for certain types of biological networks such as protein–protein interaction networks. They further propose to use the (normalized) number of paths of length three (L3) between two nodes to perform link prediction. We next present a way to approximate L3 using GLEE. This achieves good performance in those networks where CN and JS are invalid and show that GLEE can take into account higher-order connectivity of the graph.

LEMMA 3.9 Assume S is the GLEE of a graph G of dimension $d = n$. Then, the number of paths of length three between two distinct nodes i and j is

$$L3(i,j) = -\deg(i)\deg(j)\, C_{N(i)} \cdot C_{N(j)}^\top + \sum_{k \in N(i) \cap N(j)} \|s_k\|^2 . \tag{3.8}$$

Proof. The number of paths of length three between i and j is $(\mathbf{A}^3)_{ij}$, where $\mathbf{A}$ is the adjacency matrix of G. We have

$$(\mathbf{A}^3)_{ij} = \sum_{k \in N(i)} \sum_{\substack{l \in N(j) \\ l \neq k}} a_{kl} = -\sum_{k \in N(i)} \sum_{\substack{l \in N(j) \\ l \neq k}} s_k \cdot s_l^\top \tag{3.9}$$

$$= -\sum_{k\in N(i)}\sum_{l\in N(j)} s_k \cdot s_l^\top + \sum_{k\in N(i)\cap N(j)} s_k \cdot s_k^\top \tag{3.10}$$

$$= -|N(i)||N(j)|C_{N(i)} \cdot C_{N(j)}^\top + \sum_{k\in N(i)\cap N(j)} \|s_k\|^2, \tag{3.11}$$

where the last expression follows by the linearity of the dot product, and is equivalent to (3.8). □

When $d < n$, we can estimate $\deg(i)$ by $\|s_i^d\|^2$ and $N(i)$ by $\hat{N}(i)$ as before, with the help of an estimator $\hat{\theta}$ from Section 3.1.

3.3 *Runtime analysis*

On a graph G with n nodes, finding the k largest eigenvalues and eigenvectors of the Laplacian takes $O(kn^2)$ time, if one uses algorithms for fast approximate singular value decomposition [12, 13]. Given a k-dimensional embedding matrix S, reconstructing the graph is as fast as computing the product $S \cdot S^\top$ and applying the threshold θ to each entry, thus it takes $O(n^\omega + n^2)$, where ω is the exponent of matrix multiplication. Approximating the number of CN between nodes i and j depends only on the dot products between embeddings corresponding to their neighbours, thus it takes $O(k \times \min(\deg(i), \deg(j)))$, while approximating the number of paths of length 3 takes $O(k \times \deg(i) \times \deg(j))$.

4. Related work

Spectral analyses of the Laplacian matrix have multiple applications in graph theory, network science and graph mining [14–16]. Indeed, the eigendecomposition of the Laplacian has been used for sparsification [17], clustering [18], dynamics [19, 20], robustness [21, 22], etc. We here discuss those applications that are related to the general topic of this work, namely, dimensionality reduction of graphs.

One popular application is the use of Laplacian eigenvectors for graph drawing [23, 24], which can be thought of as graph embedding for the specific objective of visualization. In [24], one such method is outlined, which, similarly to GLEE, assigns a vector, or higher-dimensional position, to each node in a graph using the eigenvectors of its Laplacian matrix, in such a way that the resulting vectors have certain desirable geometric properties. However, in the case of [24], those geometric properties are externally enforced as constraints in an optimization problem, whereas GLEE uses the intrinsic geometry already present in a particular decomposition of the Laplacian. Furthermore, their method focuses on the eigenvectors corresponding to the smallest eigenvalues of the Laplacian, while GLEE uses those corresponding to the largest eigenvalues, that is, to the best approximation to the Laplacian through singular value decomposition.

On another front, many graph embedding algorithms have been proposed, see for example [25, 26] for extensive reviews. Most of these methods fall in one of the following categories: matrix factorization, random walks or deep architectures. Of special importance to us are methods that rely on matrix factorization. Among many advantages, we have at our disposal the full toolbox of spectral linear algebra to study them [27–30]. Examples in this category are the aforementioned Laplacian Eigenmaps (LE) [7, 8] and graph factorization (GF) [31]. One important difference between GLEE and LE is that LE uses the small eigenvalues of the normalized Laplacian $\mathbf{D^{-1}L}$, while GLEE uses the large eigenvalues of $\mathbf{L}$. Furthermore, LE does not present the rich geometry of the simplex. GF finds a decomposition of the weighted adjacency matrix $\mathbf{W}$ with a regularization term. Their objective is to find embeddings $\{s_i\}$ such that $s_i \cdot s_j = a_{ij}$, whereas in our case we try to reconstruct $s_i \cdot s_j = \mathbf{L_{ij}}$. This means that the embeddings found by GF will present different geometric properties. There are many other methods of dimensionality reduction on graphs that depend on matrix factorization [32–34]. However, even if some parameterization, or special case, of any of these methods results in a method resembling the singular value decomposition of the Laplacian (thus imitating GLEE), to the authors' knowledge none of these methods make direct use of its intrinsic geometry.

Among the methods based on random walks we find DeepWalk [35] and node2vec [36], both of which adapt the framework of word embeddings [37] to graphs by using random walks and optimize a shallow architecture. It is also worth mentioning NetMF [38] which unifies several methods in a single algorithm that depends on matrix factorization and thus unifies the two previous categories.

Among the methods using deep architectures, we have the deep autoencoder Structural Deep Network Embedding (SDNE) [39]. It penalizes representations of similar nodes that are far from each other using the same objective as LE. Thus, SDNE is also based on the distance-minimization approach. There is also [40] which obtains a non-linear mapping between the probabilistic mutual information (PMI) matrix of a sampled network and the embedding space. This is akin to applying the distance-minimization assumption not to the graph directly but to the PMI matrix.

Others have used geometric approaches to embedding. For example, [41] and [42] find embeddings on the surface of a sphere, while [43] and [44] use the hyperbolic plane. These methods are generally developed under the assumption that the embedding space is used to generate the network itself. They are therefore aimed at recovering the generating coordinates, and not, as in GLEE's case, at finding a general representation suitable for downstream tasks.

5. Experiments

We put into practice the procedures detailed in Sections 3.1 and 3.2 to showcase GLEE's performance in the tasks of link prediction and graph reconstruction. Code to compute the GLEE embeddings of networks and related computations is publicly available at [45]. For our experiments, we use the following baselines: GF because it is a direct factorization of the adjacency matrix, node2vec because it is regarded as a reference point among those methods based on random walks, SDNE because it aims to recover the adjacency matrix of a graph (a task GLEE excels at), NetMF because it generalizes several other well-known techniques and LE because it is the method that most directly resembles our own. In this way, we cover all of the categories explained in Section 4 and use either methods that resemble GLEE closely or methods that have been found to generalize other techniques. For node2vec and SDNE, we use default parameters. For NetMF, we use the spectral approximation with rank 256. The datasets we use are outlined in Table 1. Beside comparing GLEE to the other algorithms, we are interested in how the graph's structure affects performance of each method. This is why we have chosen datasets have similar number of nodes and edges, but different values of average clustering coefficient. Accordingly, we report our results with respect to the average clustering coefficient of each dataset and the number of dimensions of the embedding (the only parameter of GLEE). In Appendix B, we compare the performance of each estimator explained in Section 3.1. In the following experiments, we use $\hat{\theta}_k$ as our estimator for θ_{opt}.

5.1 *Graph reconstruction*

Given a GLEE matrix S^d, how well can we reconstruct the original graph? This is the task of graph reconstruction. We use as performance metric the *precision at k* measure, defined as the precision of the first k reconstructed edges. Note that *precision at k* must always decrease when k grows large, as there will be few correct edges left to reconstruct.

TABLE 1 *Datasets used in this work (all undirected, unweighted): number of nodes n, number of edges m and average clustering coefficient $\bar{c}$ of the largest connected component of each network. AS, autonomous systems of the Internet.*

Name	n	m	$\bar{c}$	Type
PPI [46]	4,182	13,343	0.04	Protein interaction
wiki-vote [47]	7,066	100,736	0.14	Endorsement
caida [47]	26,475	53,381	0.21	AS Internet
CA-HepTh [48]	8,638	4,806	0.48	Co-authorship
CA-GrQc [48]	4,158	13,422	0.56	Co-authorship

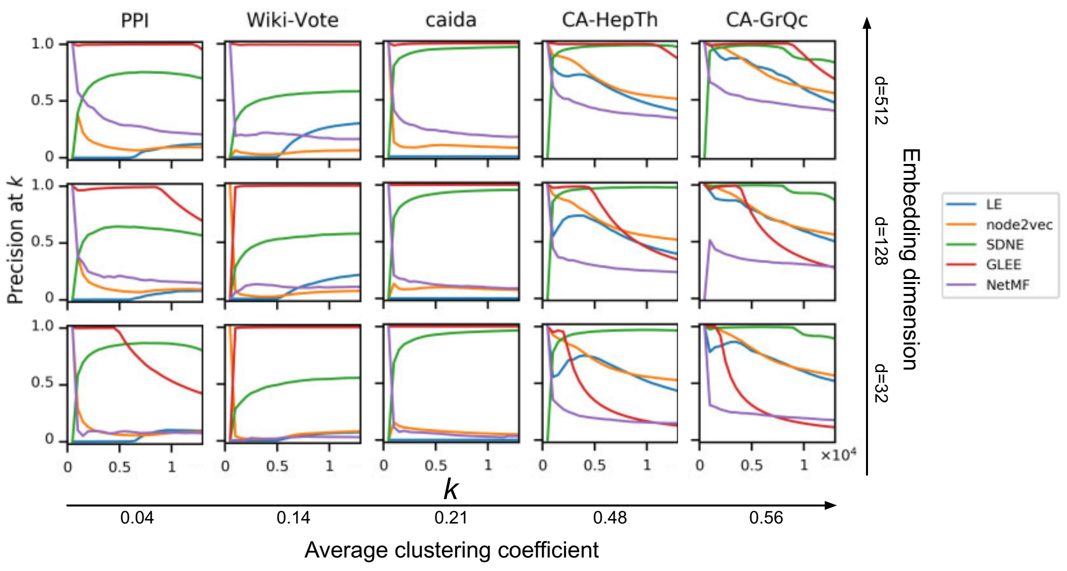

FIG. 2. Graph reconstruction. GLEE performs best in networks with low clustering coefficient, presumably because it depends on the large eigenvalues of the Laplacian, which encode micro-level graph structure.

Following Section 3.1, we reconstruct the edge (i,j) if $s_i^d \cdot s_j^d < \hat{\theta}$. The further the dot product is from 0 (the ideal value for non-edges), the more confident we are in the existence of this edge. For LE, we reconstruct the edge (i,j) according to how small the distance between their embeddings is. For both GF, node2vec and NetMF, we reconstruct edges based on how high their dot product is. SDNE is a deep autoencoder and thus its very architecture involves a mechanism to reconstruct the adjacency matrix of the input graph.

We show results in Fig. 2, where we have ordered datasets from left to right in ascending order of clustering coefficient, and from bottom up in ascending order of embedding dimension. GF results omitted from this figure as it scored close to 0 for all values of k and d. On CA-GrQc, for low embedding dimension $d = 32$, SDNE performs best among all methods, followed by node2vec and LE. However, as d increases, GLEE substantially outperforms all others, reaching an almost perfect precision score at the

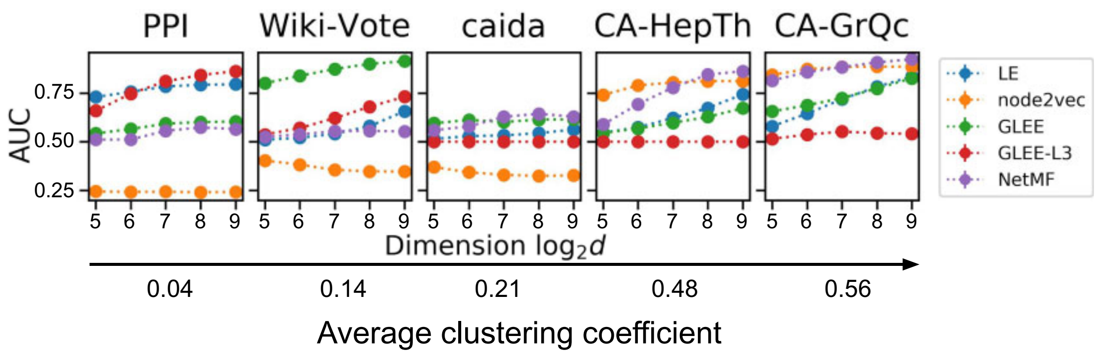

FIG. 3. Link prediction results. We approximate the number of CN (GLEE), or the number of paths of length 3 (GLEE-L3). Each circle is the average of 10 realizations; error bars too small to show at this scale. GF and SDNE perform close to 0.5 and 0.6 independently of d or dataset (not shown). Datasets ordered from left to right in increasing order of clustering.

first 10,000 reconstructed edges. Interestingly, other methods do not substantially improve performance as d increases. This analysis is also valid for `CA-HepTh`, another dataset with high clustering coefficient. However, on `PPI`, our dataset with lowest clustering coefficient, GLEE drastically outperforms all other methods for all values of d. Interestingly, LE and node2vec perform well compared to other methods in datasets with high clustering, but their performance drops to near zero on `PPI`. We hypothesize that this is due to the fact that LE and node2vec depend on the 'community-aware' assumption, thereby assuming that two proteins in the same cluster would interact with each other. This is the exact point that [11] refutes. On the other hand, GLEE directly encodes graph structure, making no assumptions about the original graph, and its performance depends more directly on the embedding dimension than on the clustering coefficient, or on any other assumption about graph structure. GLEE's performance on datasets `PPI`, `Wiki-Vote` and `caida` point to the excellent potential of our method in the case of low clustering coefficient.

5.2 *Link prediction*

Given the embedding of a large subgraph of some graph G, can we identify which edges are missing? The experimental setup is as follows. Given a graph G with n nodes, node set V and edge set E_{obs}, we randomly split its edges into train and test sets E_{train} and E_{test}. We use $|E_{\text{train}}| = 0.75n$, and we make sure that the subgraph induced by E_{train}, denoted by G_{train}, is connected and contains every node of V. We then proceed to compute the GLEE of G_{train} and test on E_{test}. We report area under receiver operator curve (AUC) metric for this task. We use both techniques described in Sections 3.2.1 and 3.2.2, which we label GLEE and GLEE-L3, respectively.

Figure 3 shows that node2vec repeats the behaviour seen in graph reconstruction of increased performance as clustering coefficient increases, though again it is fairly constant with respect to embedding dimension. This observation is also true for NetMF. On the high clustering datasets, LE and GLEE have comparable performance to each other. However, either GLEE or GLEE-L3 perform better than all others on the low clustering datasets `PPI`, `Wiki-Vote`, as expected. Also as expected, the performance of GLEE-L3 decreases as average clustering increases. Note that GLEE and LE generally improve performance when d increases, whereas node2vec and SDNE do not improve. (GF and SDNE not shown in Fig. 3 for clarity. They scored close to 0.5 and 0.6 in all datasets independently of d.) The reason why none of the methods studied here perform better than 0.6 AUC in the `caida` dataset is an open question left for future research. We conclude that the hybrid approach of NetMF is ideal for high clustering coefficient, whereas GLEE is a viable option in the case of low clustering coefficient as evidenced by the results on `PPI`, `Wiki-Vote` and `caida`.

6. Conclusions

In this work, we have presented the GLEE, a geometric approach to graph embedding that exploits the intrinsic geometry of the Laplacian. When compared to other methods, we find that GLEE performs the best when the underlying graph has low clustering coefficient, while still performing comparably to other state-of-the-art methods when the clustering coefficient is high. We hypothesize that this is due to the fact that the large eigenvalues of the Laplacian correspond to the small eigenvalues of the adjacency matrix and thus represent the structure of the graph at a micro-level. Furthermore, we find that GLEE's performance increases as the embedding dimension increases, something we do not see in other methods. In contrast to techniques based on neural networks, which have many hyperparameters and costly training phases, GLEE has only one parameter other than the embedding dimension, the threshold θ, and we have provided three different ways of optimizing for it. Indeed, GLEE only depends on the singular value decomposition of the Laplacian matrix.

We attribute these desirable properties of GLEE to the fact that it departs from the traditional literature of graph embedding by replacing the 'community aware' notion (similar nodes' embeddings must be similar) with the notion of directly encoding graph structure using the geometry of the embedding space. In all, we find that GLEE is a promising alternative for graph embedding due to its simplicity in both theoretical background and computational implementation, especially in the case of low clustering coefficient. By taking a direct geometric encoding of graph structure using the simplex geometry, GLEE covers the gap left open by the 'community-aware' assumption of other embedding techniques, which requires high clustering. Future lines of work will explore what other geometric properties of the embedding space can yield interesting insight, as well as what are the important structural properties of graphs, such as clustering coefficient, that affect the performance of these methods.

Funding

National Science Foundation (IIS-1741197); and by the Combat Capabilities Development Command Army Research Laboratory and was accomplished (Cooperative Agreement Number W911NF-13-2-0045; U.S. Army Research Lab Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright notation here on.

A. Threshold estimators

We present two other estimators of θ_{opt} to accompany the heuristic $\hat{\theta}_c = -0.5$ mentioned in Section 3.1.

A.1 *Kernel density estimation*

As can be seen in Fig. A.1, the problem of finding a value of θ that sufficiently separates the peaks corresponding to edges (around the peak centred at -1) and non-edges (around the peak centred at 0) can be stated in terms of density estimation. That is, given the histogram of values of $s_i \cdot s_j^\top$ for all i, j, we can approximate the density of this empirical distribution by some density function f_k. A good heuristic

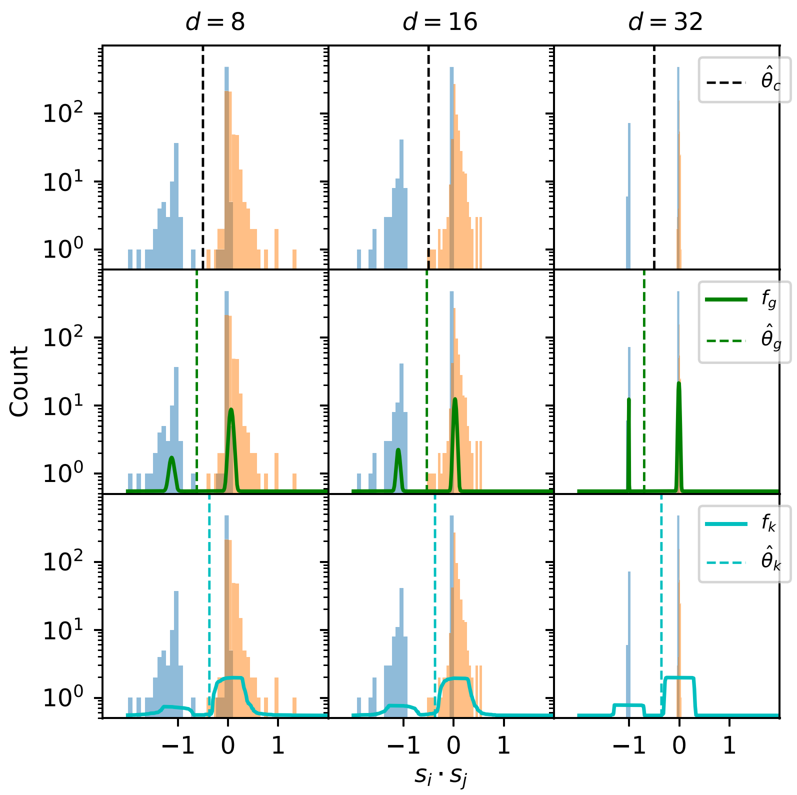

FIG. A.1. Distribution of dot products $\{s_i \cdot s_j^\top\}$ in the Karate Club graph. Columns show different dimension d. Dot products of existing edges are in blue. Non-edges are in orange. Each row shows a different estimator of θ_{opt}. Top row shows the constant value $\hat{\theta}_c = -0.5$. Middle row shows our GMM estimator $\hat{\theta}_g$ (Algorithm 1). Bottom row shows our Kernel Density Estimator $\hat{\theta}_k$ (Section A.1).

estimator of θ_{opt} is the value that minimizes f_k between the peaks near -1 and 0. For this purpose, we use Kernel Density Estimation over the distribution of $s_i \cdot s_j^\top$ and a box kernel (a.k.a. 'top hat' kernel) function to define

$$f_k(x) \propto \sum_{i<j}^{n} 1\{x - s_i \cdot s_j^\top < h\}. \tag{A.1}$$

We then use gradient descent to find the minimal value of f_k between the values of -1 and 0. We call this value $\hat{\theta}_k$. We have found experimentally that a value of $h = 0.3$ gives excellent results, achieving near zero error in the reconstruction task (Fig. A.1, middle row).

A.2 *Gaussian mixture models*

Here, we use a GMM over the distribution of $s_i \cdot s_j$. The model will find the two peaks near -1 and 0 and fit each to a Gaussian distribution. Once the densities of said Gaussians have been found, say f_1 and f_2, we define the estimator $\hat{\theta}_g$ as that point at which the densities are equal (see Fig. A.1, bottom row).

However, we found that a direct application of this method yields poor results due to the sparsity of network datasets. High sparsity implies that the peak at 0 is orders of magnitude higher than the one at -1. Thus, the left peak will usually be hidden by the tail of the right one so that the GMM cannot detect it. To solve this issue we take two steps. First, we use a Bayesian version of GMM that accepts priors for the Gaussian means and other parameters. This guides the GMM optimization algorithm to find the right peaks at the right places. Second, we sub-sample the distribution of dot products in order to minimize the difference between the peaks, and then to fix it back after the fit. Concretely, put $r = \sum_{i<j} 1\{s_i \cdot s_j^\top < \hat{\theta}_c\}$. That is, r is the number of dot products less than the constant $\hat{\theta}_c = -0.5$. Instead of fitting the GMM to all the observed dot products, we fit it to the set of all r dot products less than $\hat{\theta}_c$ plus a random sample of r dot products larger than $\hat{\theta}_c$. This temporarily fixes the class imbalance, which we recover after the model has been fit as follows. The GMM fit will yield a density for the sub-sample as $f_g = w_1 f_1 + w_2 f_2$, where f_i is the density of the ith Gaussian, and w_i are the mixture weights, for $i = 1, 2$. Since we sub-sampled the distribution, we will get $w_1 \approx w_2 \approx 0.5$, but we need the weights to reflect the original class imbalance. For this purpose, we define $\hat{w}_1 = \hat{m}/\binom{n}{2}$ and $\hat{w}_2 = 1 - \hat{w}_1$, where $\hat{m}$ is an estimate for the number of edges in the graph. (This can be estimated in a number of ways, for example one may put $\hat{m} = r$, or $\hat{m} = n\log(n)$.) Finally, we define the estimator as the value that satisfies

$$\hat{w}_1 f_1(\hat{\theta}_g) = \hat{w}_2 f_2(\hat{\theta}_g), \tag{A.2}$$

under the constraint that $-1 < \hat{\theta}_g < 0$. Since f_1 and f_2 are known Gaussian densities, Equation A.2 can be solved analytically.

In this case, due to sparsity, the problem of optimizing the GMM is one of non-parametric density estimation with extreme class imbalance. We solve it by utilizing priors for the optimization algorithm, as well as sub-sampling the distribution of dot products, according to some of its known features (i.e. the fact that the peaks will be found near -1 and 0), and we account for the class imbalance by estimating graph sparsity separately. Finally, we define the estimator $\hat{\theta}_g$ according to Equation A.2. Algorithm 1 gives an overview of this procedure. For a comparison between the effectiveness of the three different estimators $\hat{\theta}_c, \hat{\theta}_k, \hat{\theta}_g$, see Appendix B.

Algorithm 1 Estimating θ_{opt} with a Gaussian Mixture Model

```
1. procedure GMM($\{s_i\}_{i=1}^n, \hat{\theta}_c, \hat{m}$)
2.     $L \leftarrow \{s_i \cdot s_j^\top : s_i \cdot s_j^\top < \hat{\theta}_c\}$
3.     $R \leftarrow$ random sample of size $|L|$ of $\{s_i \cdot s_j^\top : s_i \cdot s_j^\top \geq \hat{\theta}_c\}$
4.     $w_1, w_2, f_1, f_2 \leftarrow$ fit a Bayesian GMM to $L \cup R$
5.     $\hat{w}_1 \leftarrow \hat{m}/\binom{n}{2}$
6.     $\hat{w}_2 \leftarrow 1 - \hat{w}_1$
7.     $\hat{\theta}_g \leftarrow$ solution of $\hat{w}_1 f_1(\theta) = \hat{w}_2 f_2(\theta)$
8.     return $\hat{\theta}_g$
9. end procedure
```

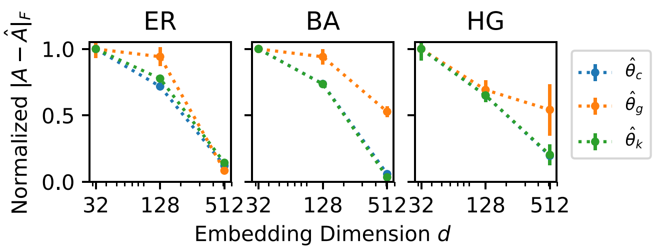

FIG. B.1. Estimator comparison. We compute the three different estimators on three different random graph models: ER, BA and HG. All graphs have $n = 10^3$ nodes, and average degree $\langle k \rangle \approx 8$. Hyperbolic graphs generated with degree distribution exponent $\gamma = 2.3$. We show the average of 20 experiments; error bars mark two standard deviations. Values normalized in the range [0, 1].

B. Estimator comparison

In Section 3.1 and Appendix A, we outlined three different schemes to estimate θ_{opt} which resulted in $\hat{\theta}_c, \hat{\theta}_k, \hat{\theta}_g$. *Which one is the best?* We test each of these estimators on three random graph models: Erdös–Rényi (ER) [49], Barabási–Albert (BA) [50] and Hyperbolic Graphs (HG) [51]. For each random graph with adjacency matrix $\mathbf{A}$, we compute the Frobenius norm of the difference between the reconstructed adjacency matrix $\hat{\mathbf{A}}$ using each of the three estimators. In Fig. B.1, we show our results. We see that $\hat{\theta}_c$ and $\hat{\theta}_k$ achieve similar performance across datasets, while $\hat{\theta}_g$ outperforms the other two for ER at $d = 512$, though it has high variability in the other models. From these results, we conclude that at low dimensions $d = 32$, too much information has been lost and thus there is no hope to learn a value of $\hat{\theta}$ that outperforms the heuristic $\hat{\theta}_c = -0.5$. However, at larger dimensions, the estimators $\hat{\theta}_g$ and $\hat{\theta}_k$ perform better, with different degrees of variability. We conclude also that no single heuristic for $\hat{\theta}$ is best for all types of graphs. In the rest of our experiments, we use $\hat{\theta}_k$ as our estimator for θ_{opt}. We highlight that even though θ_k is better than θ_c in some datasets, it might be costly to compute, while θ_c incurs no additional costs.

REFERENCES

1. CHEN, H., PEROZZI, B., AL-RFOU, R. & SKIENA, S. (2018) A tutorial on network embeddings. Preprint. arXiv:1808.02590.

2. CHEN, S., NIU, S., AKOGLU, L., KOVACEVIC, J. & FALOUTSOS, C. (2017) Fast, warped graph embedding: unifying framework and one-click algorithm. *CoRR*, abs/1702.05764.

3. SRINIVASAN, B. & RIBEIRO, B. (2019) On the equivalence between node embeddings and structural graph representations. *CoRR*, abs/1910.00452.

4. JIN, D., ROSSI, R. A., KOH, E., KIM, S., RAO, A. & KOUTRA, D. (2019) Latent network summarization: bridging network embedding and summarization. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019* (A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi & G. Karypis eds). New York: ACM, pp. 987–997.

5. DEVRIENDT, K. & VAN MIEGHEM, P. (2019) The simplex geometry of graphs. *J. Compl. Netw.*

6. FIEDLER, M. (2011) *Matrices and Graphs in Geometry*, vol. 139 of Encyclopedia of Mathematics and its Applications. Cambridge: Cambridge University Press.

7. BELKIN, M. & NIYOGI, P. (2002) Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems 14, NIPS 2014* (T. D. Diettereich, S. Becker & Z. Ghahramani eds). Cambridge, MA: MIT Press, pp. 585–591.

8. BELKIN, M. & NIYOGI, P. (2003) Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, **15**, 1373–1396.

9. Zachary, W. W. (1977) An information flow model for conflict and fission in small groups. *J. Anthropol. Res.*, **33**, 452–473.

10. Sarkar, P., Chakrabarti, D. & Moore, A. W. (2011) Theoretical justification of popular link prediction heuristics. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011* (T. Walsh ed). Palo Alto, CA: IJCAI/AAAI, pp. 2722–2727.

11. Kovács, I. A., Luck, K., Spirohn, K., Wang, Y., Pollis, C., Schlabach, S., Bian, W., Kim, D.-K., Kishore, N., Hao, T., Calderwood, M. A., Vidal, M. & Barabási, A.-L. (2019) Network-based prediction of protein interactions. *Nat. Commun.*, **10**, 1–8.

12. Halko, N., Martinsson, P.-G. & Tropp, J. A. (2011) Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, **53**, 217–288.

13. Trefethen, L. N. & Bau III, D. (1997) *Numerical Linear Algebra*, vol. 50. Philadelphia, PA: SIAM.

14. Newman, M.E.J. (2018) *Networks*. Oxford: Oxford University Press.

15. Spielman, D. (2017) Graphs, vectors, and matrices. *Bull. Am. Math. Soc.*, **54**, 45–61.

16. Van Mieghem, P. (2010) *Graph Spectra for Complex Networks*. Cambridge: Cambridge University Press.

17. Spielman, D. & Srivastava, N. (2011) Graph sparsification by effective resistances. *SIAM J. Comput.*, **40**, 1913–1926.

18. Von Luxburg, U (2007) A tutorial on spectral clustering. *Stat. Comput.*, **17**, 395–416.

19. Prakash, B., Vreeken, J. & Faloutsos, C. (2014) Efficiently spotting the starting points of an epidemic in a large graph. *Knowl. Inform. Syst.*, **38**, 35–59.

20. Van Mieghem, P., Sahnehz, F. D. & Scoglioz, C. (2014) An upper bound for the epidemic threshold in exact Markovian SIR and SIS epidemics on networks. *53rd IEEE Conference on Decision and Control, CDC 2014*. Piscataway, NJ: IEEE.

21. Jamakovic, A. & Van Mieghem, P. (2008) On the robustness of complex networks by using the algebraic connectivity. *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet: 7th International IFIP-TC6 Networking Conference* (A. Das, H. K. Pung, F. B.-S. Lee & L. W.-C. Wong eds). New York, NY: Springer.

22. Shahrivar, E.M., Pirani, M. & Sundaram, S. (2015) Robustness and algebraic connectivity of random interdependent networks. *IFAC-PapersOnLine*, **48**, 252–257.

23. Koren, Y. (2005) Drawing graphs by eigenvectors: theory and practice. *Comput. Math. Appl.*, **49**, 1867–1888.

24. Pisanski, T. & Shawe-Taylor, J. (2000) Characterizing graph drawing with eigenvectors. *J. Chem. Inform. Comput. Sci.*, **40**, 567–571.

25. Goyal, P. & Ferrara, E. (2018) Graph embedding techniques, applications, and performance: a survey. *Knowl.-Based Syst.*, **151**, 78–94.

26. Hamilton, W. L., Ying, R. & Leskovec, J. (2017) Representation learning on graphs: methods and applications. *IEEE Data Eng. Bull.*, **40**, 52–74.

27. Charisopoulos, V., Benson, A. R. & Damle, A. (2019) Incrementally updated spectral embeddings. *CoRR*, abs/1909.01188.

28. Chen, C. & Tong, H. (2015) Fast eigen-functions tracking on dynamic graphs. *Proceedings of the 2015 SIAM International Conference on Data Mining, SDM 2015* (S. Venkatasubramanian & J. Ye eds). Philadelphia, PA: SIAM, pp. 559–567.

29. Chen, C. & Tong, H.(2017) On the eigen-functions of dynamic graphs: fast tracking and attribution algorithms. *Stat. Anal. Data Mining*, **10**, 121–135.

30. Levin, K., Roosta-Khorasani, F., Mahoney, M. W. & Priebe, C.E. (2018) Out-of-sample extension of graph adjacency spectral embedding. *Proceedings of the 35th International Conference on Machine Learning, ICML 2018* (J. G. Dy & A. Krause eds), vol. 80. New York , NY: ACM, vol. 80. pp. 2981–2990.

31. Ahmed, A., Shervashidze, N., Narayanamurthy, S. M., Josifovski, V. & Smola, A. J. (2013) Distributed large-scale natural graph factorization. *22nd International Conference World Wide Web Conference, WWW 2013* (D. Scwabe, V. A. F. Almeida, H. Glaser, R, Baeza-Yates & S. B. Mon eds). New York, NY: ACM, pp. 37–48.

32. Cai, D., He, X., Han, J. & Huang, T. S. (2011) Graph regularized nonnegative matrix factorization for data representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, **33**, 1548–1560.

33. Kuang, D., Park, H. & Ding, C. H. Q. (2012) Symmetric nonnegative matrix factorization for graph clustering. *SDM. Proceedings of the 12th SIAM International Conference on Data Mining, SDM 2012*. Philadelphia, PA: SIAM, pp. 106–117.

34. Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W. & Yang, S. (2017) Community preserving network embedding. *AAAI. Proceedings of the 31st AAAI Conference on Artificial Intelligence* (S. P. Singh & S. Markovitch eds). Palo Alto, CA: AAAI Press, pp. 203–209.

35. Perozzi, B., Al-Rfou, R. & Skiena, S. (2014) DeepWalk: online learning of social representations. *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2014* (S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang & R. Gnani eds). New York, NY: ACM, pp. 701–710.

36. Grover, A. & Leskovec, J. (2016) node2vec: scalable feature learning for networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2016* (B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen & R. Rastogi eds). New York, NY: ACM, pp. 855–864.

37. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. (2013) Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, (C. J. C. Burges, L. Bottou, Z. Ghahramani & K. Q. Weinberger eds). Red Hook, NY: Curran Associates, Inc., pp. 3111–3119.

38. Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K. & Tang, J. (2018) Network embedding as matrix factorization: unifying DeepWalk, LINE, PTE, and node2vec. *WSDM*. ACM, pp. 459–467.

39. Wang, D., Cui, P. & Zhu, W. (2016) Structural deep network embedding. *KDD*. pp. 1225–1234.

40. Cao, S., Lu, W. & Xu, Q. (2016) Deep neural networks for learning graph representations. *Proceedings of the 13th AAAI Conference on Artificial Intelligence* (D. Schuurmans & M. P. Wellman eds). Palo Alto, CA: AAAI, pp. 1145–1152.

41. Estrada, E., Sánchez-Lirola, M. G. & de la Peña, J. A. (2014) Hyperspherical embedding of graphs and networks in communicability spaces. *Discrete Appl. Math.*, **176**, 53–77.

42. Pereda, M. & Estrada, E. (2019) Machine learning analysis of complex networks in hyperspherical space. *Patt. Recogn.*, **86**, 320–331.

43. Papadopoulos, F., Kitsak, M., Serrano, M. Á., Boguná, M. & Krioukov, D. (2012) Popularity versus similarity in growing networks. *Nature*, **489**, 537.

44. Nickel, M. & Kiela, D. (2018) Learning continuous hierarchies in the Lorentz model of hyperbolic geometry. *Proceedings of the 35th International Conference on Machine Learning, ICML 2018* (J. G. Dy & A. Krause eds), vol. 80, New York, NY: ACM, pp. 3776–3785.

45. Torres, L. (2019) GLEE: Geometric Laplacian Eigenmap Embedding. github.com/leotrs/glee.

46. Rolland, T., Taşan, M., Charloteaux, B., Pevzner, S. J., Zhong, Q., Sahni, N., Yi, S., Lemmens, I., Fontanillo, C., Mosca, R. et al. (2014) A proteome-scale map of the human interactome network. *Cell*, **159**, 1212–1226.

47. Leskovec, J., Huttenlocher, D. P. & Kleinberg, J. M. (2010) Predicting positive and negative links in online social networks. *Proceedings of the 19th International Conference on World Wide Web, WWW 2010* (M. Rappa, P. Jones, J. Freire & S. Chakrabarti eds). New York, NY: ACM, pp. 641–650.

48. Leskovec, J., Kleinberg, J. M. & Faloutsos, C. (2007) Graph evolution: densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, **1**, 2.

49. Erdös, P. & Rényi, A. (1960) On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, **5**, 17.

50. Barabási, A.-L. & Albert, R. (1999) Emergence of scaling in random networks. *Science*, **286**, 509–512.

51. Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A. & Boguñá, M. (2010) Hyperbolic geometry of complex networks. *Phys. Rev. E*, **82**, 036106.