GAGE: Geometry Preserving Attributed Graph Embeddings

Charilaos I. Kanatsoulis
University of Pennsylvania
Philadelphia, Pennsylvania, USA
kanac@seas.upenn.edu

Nicholas D. Sidiropoulos
University of Virginia
Charlottesville, Virginia, USA
nikos@virginia.edu

ABSTRACT

Node embedding is the task of extracting concise and informative representations of certain entities that are connected in a network. Various real-world networks include information about both node connectivity and certain node attributes, in the form of features or time-series data. Modern representation learning techniques employ both the connectivity and attribute information of the nodes to produce embeddings in an unsupervised manner. In this context, deriving embeddings that preserve the geometry of the network and the attribute vectors would be highly desirable, as they would reflect both the topological neighborhood structure and proximity in feature space. While this is fairly straightforward to maintain when only observing the connectivity or attribute information of the network, preserving the geometry of both types of information is challenging. A novel tensor factorization approach for node embedding in attributed networks is proposed in this paper, that preserves the distances of both the connections and the attributes. Furthermore, an effective and lightweight algorithm is developed to tackle the learning task and judicious experiments with multiple state-of-the-art baselines suggest that the proposed algorithm offers significant performance improvements in downstream tasks.

CCS CONCEPTS

• **Information systems → Social networks**; **Web mining**; • **Computing methodologies → Learning latent representations**; • **Networks → Network algorithms**.

KEYWORDS

networks, graphs, tensors, representation learning, embedding, multi dimensional scaling

ACM Reference Format:
Charilaos I. Kanatsoulis and Nicholas D. Sidiropoulos. 2022. GAGE: Geometry Preserving Attributed Graph Embeddings. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3488560.3498467

WSDM '22, February 21–25, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9132-0/22/02…$15.00
https://doi.org/10.1145/3488560.3498467

1 INTRODUCTION

Network science studies the behavior of entities, belonging to one or more communities, via observing their mutual interactions [3]. Networks and network science have attracted considerable attention in science and engineering, since they offer an elegant abstraction of various physical, social, and engineered systems – and effective tools to analyze them [12, 25]. Networks are nowadays ubiquitous in a plethora of science and engineering disciplines, including social, communication, and biological networks, to name a few.

Networks are usually represented by graphs, which are informative abstractions and model the interactions in the system. In particular, graph representations encode the connectivity information of different entities (nodes) through a set of edges. The connectivity information in a network is important and describes each node in the network with respect to the rest of the nodes. In real world networks, the entities are not only defined by their connectivity with other entities, but can also be described by a set of measurements or attributes, which offer a node characterisation at an individual level, and are usually very informative. Although graphs offer an elegant representation of the network entities, individual representations of the entities is also required that is not necessarily described by relations with respect to subsets of the community. Furthermore, when attributes are available for each node, which is often the case in practice, it is essential to combine both connectivity and attribute information in a single, universal representation, that encapsulates as much information as possible. Moreover, a variety of interesting networks involve millions of nodes, which makes graph representation of nodes highly impractical for certain tasks.

The aforementioned challenges underscore the need for concise and informative representation of network nodes that is conducive for exploratory analysis, as well as downstream applications. This has motivated a considerable body of research on embedding graph nodes in a low-dimensional vector space, using graph and attribute information in an unsupervised manner. The task is also known as unsupervised node or graph representation learning. The objective of unsupervised node embedding is twofold. On the one hand, the embeddings should capture the maximum amount of knowledge present in the graph and attributes so that information loss is avoided. Towards this end, a key to successful node embeddings is to be able to preserve the geometry of the network, defined by proximity in both the connectivity and the attributes of the nodes. On the other hand the embedding should be able to boost the performance of various downstream network tasks, such as node classification, link prediction, and community detection, to name a few. Concise node representations produced by embedding algorithms can significantly benefit feature-based tools such as logistic regression, support vector machines, and even neural networks – especially when only limited training data are accessible.

A plethora of methods have been proposed to perform node embedding. Early works approached the node representation learning task using only the connectivity information of the network. A number of them focused on properly defining a similarity measure on the connectivity information and performing matrix factorization on it [1, 4, 5, 26, 29, 32, 34, 36, 39]. The work in [20] performs coupled tensor-matrix factorization to learn representations in the context of knowledge graphs. Random walks have also been successfully employed to generate node embeddings, e.g., [16, 27]. More recently, the focus of research has shifted towards generating embeddings for attributed networks. The work in [39] generalizes deepwalk [27] to the case where attributes are available, while [19] performs label-informed attributed node representation learning in a semi-supervised setting. Neural networks for network science tasks have also gained significant attention lately. In particular, graph convolutional neural networks and graph auto-encoders are very popular for attributed node embedding [6, 10, 13, 21, 22, 37, 38]. Works have also been proposed to perform inductive embedding, e.g., [17, 37] where a graph convolutional network is trained with multiple graphs. Finally, the work in [2] employs a tensor decomposition model and jointly factors the conventional adjacency along with a k-nearest neighbor matrix of the attributes.

Our work is motivated by the following question: *Can we produce node embeddings such that we provably preserve the geometry of 1) the distances associated with the connectivity information of the network, and 2) the distances associated with the attributed information of the network, in an unsupervised manner?* This is a well motivated problem, since maintaining the network geometry is a fundamental objective of representation learning, and, as we will show in this paper, doing so significantly improves the performance of several downstream tasks. The problem can be informally stated as follows:

- **Given:** the connectivity and attribute information of network nodes.
- **Produce:** Low dimensional node representations that preserve both the connectivity and attribute geometry.

We propose Geometry-preserving Attributed Graph Embedding (GAGE) – a principled approach to extract node embeddings in an unsupervised fashion. GAGE enjoys several favorable properties.

- By design, the produced embeddings preserve node geometry, as inferred from both the node adjacency matrix and the node attributes.
- The node embeddings are unique and thus permutation invariant, meaning that any reordering of the nodes in the adjacency representation yield the same embeddings.
- The approach is applicable to both undirected and directed networks.
- The proposed approach is flexible and does not require connectivity and attribute information for every node; embeddings can be produced for nodes with partially/completely missing connectivity *or* attribute information (but not both).
- The proposed algorithm is lightweight and scalable – it can efficiently handle large networks.

To assess the performance of GAGE, we used three real attributed network benchmarks. Experimental results show that GAGE shows great promise in both tasks, markedly outperforming the baselines. The contributions of our work can be summarized as follows:

- **Novel problem formulation**: Previous work in this area hasn't formalized the intuitive requirement that the embedding should be capable of (approximately) reproducing the distances associated with the connectivity and attribute information.
- **Analysis:** We show that by leveraging the favorable properties of tensor factorization and multi dimensional scaling the proposed embedding can (approximately) reproduce both the connectivity and attribute distances.
- **Algorithm:** We propose a novel tensor factorization algorithm to perform unsupervised embedding task. The algorithm exploits the special structure of the tensor, is fast and scalable for big networks.
- **Experimental verification:** The proposed approach is assessed under node classification and link prediction settings and exhibits very promising results in both tasks.

Reproducibility: The datasets we use are all publicly available; Code and data can be found in the following link [1].
Notation: Our notation is summarized in Table 1.

Table 1: Overview of notation.

$\mathcal{V}$	$\triangleq$	Set of nodes
$\mathcal{E}$	$\triangleq$	Set of edges
$\mathcal{S}_{\mathcal{G}}$	$\triangleq$	$N \times N$ adjacency matrix
$\mathcal{A}$	$\triangleq$	$N \times d$ matrix of node attributes
$\mathbf{E}$	$\triangleq$	$N \times F$ matrix of embeddings
$\mathbf{e}_i$	$\triangleq$	$F \times 1$ mbedding vector of node v_i
a	$\triangleq$	scalar
$\boldsymbol{a}$	$\triangleq$	vector
$\boldsymbol{A}$	$\triangleq$	matrix
$\underline{\boldsymbol{A}}$	$\triangleq$	tensor
$\boldsymbol{A}_k$	$\triangleq$	k-th frontal slab of tensor $\underline{\boldsymbol{A}}$
$\boldsymbol{A}^T$	$\triangleq$	transpose of matrix $\boldsymbol{A}$
$\|\boldsymbol{A}\|_F$	$\triangleq$	Frobenius norm of matrix $\boldsymbol{A}$
$\otimes$	$\triangleq$	Kronecker product of two matrices
$\odot$	$\triangleq$	Khatri-Rao (columnwise Kronecker) product
$\lfloor x \rfloor$	$\triangleq$	largest integer that is less than or equal to x
$\boldsymbol{I}$	$\triangleq$	Identity matrix
$\mathbf{1}$	$\triangleq$	vector of ones

2 PRELIMINARIES

Before moving into the core of the paper, we briefly discuss some tensor algebra preliminaries to facilitate the exposition. The reader is referred to [23, 33] for further background on tensors.

A third-order tensor $\underline{\boldsymbol{X}} \in \mathbb{R}^{I \times J \times K}$ is a three-way with elements $\underline{\boldsymbol{X}}(i, j, k)$. It comprises three modes – columns $\underline{\boldsymbol{X}}(i, :, k)$ (: stands for $\{1, \cdots, \text{end}\}$, where end $= J$ here), rows $\underline{\boldsymbol{X}}(:, j, k)$, and fibers $\underline{\boldsymbol{X}}(i, j, :)$; and three types of slabs – horizontal $\underline{\boldsymbol{X}}(i, :, :)$, vertical $\underline{\boldsymbol{X}}(:, j, :)$ and frontal $\underline{\boldsymbol{X}}(:, :, k)$. A rank-one tensor $\underline{\boldsymbol{Z}} \in \mathbb{R}^{I \times J \times K}$ is the outer product of three vectors, $\boldsymbol{a} \in \mathbb{R}^I$, $\boldsymbol{b} \in \mathbb{R}^J$, $\boldsymbol{c} \in \mathbb{R}^K$, denoted as $\underline{\boldsymbol{Z}} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c}$, where $\circ$ is the outer product operator.

Any third order tensor can be decomposed into a sum of three way outer products (rank one tensors), i.e.

$$\underline{\boldsymbol{X}}(i, j, k) = \sum_{f=1}^{F} \boldsymbol{A}(i, f)\boldsymbol{B}(j, f)\boldsymbol{C}(k, f), \tag{1}$$

[1] https://github.com/marhar19/GAGE-Geometry-Preserving-Attributed-Graph-Embeddings

where $\boldsymbol{A} = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_F] \in \mathbb{R}^{I \times F}$, $\boldsymbol{B} = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_F] \in \mathbb{R}^{J \times F}$, $\boldsymbol{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_F] \in \mathbb{R}^{K \times F}$ are the low-rank matrix factors. The minimum F needed to synthesize $\underline{\boldsymbol{X}}$, is the *rank* of tensor $\underline{\boldsymbol{X}}$ and the corresponding decomposition is known as the *canonical polyadic decomposition* (CPD) of $\underline{\boldsymbol{X}}$ [18], denoted as $\underline{\boldsymbol{X}} = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$. A striking property, that differentiates tensors from matrices, is that the CPD of a tensor is essentially unique under mild conditions, even if the rank is higher than the dimensions. A generic result on the uniqueness of the CPD follows.

Theorem 1. *[8, p. 1019-1021] Let $\underline{\boldsymbol{X}} = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$ with $\boldsymbol{A} : I \times F$, $\boldsymbol{B} : J \times F$, and $\boldsymbol{C} : K \times F$. Assume that $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are drawn from an absolutely continuous joint distribution with respect to the Lebesgue measure in $\mathbb{R}^{(I+J+K)F}$. Also assume $I \geq J \geq K$ without loss of generality. If $F \leq 2^{\lfloor \log_2 J \rfloor + \lfloor \log_2 K \rfloor - 2}$, then the decomposition of $\underline{\boldsymbol{X}}$ in terms of $\boldsymbol{A}, \boldsymbol{B}$, and $\boldsymbol{C}$ is essentially unique, almost surely.*

Here, essential uniqueness means that if $\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}, \tilde{\boldsymbol{C}}$ also satisfy $\underline{\boldsymbol{X}} = [\![\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}, \tilde{\boldsymbol{C}}]\!]$, then $\boldsymbol{A} = \tilde{\boldsymbol{A}}\boldsymbol{\Pi}\boldsymbol{\Lambda}_1$, $\boldsymbol{B} = \tilde{\boldsymbol{B}}\boldsymbol{\Pi}\boldsymbol{\Lambda}_2$, and $\boldsymbol{C} = \tilde{\boldsymbol{C}}\boldsymbol{\Pi}\boldsymbol{\Lambda}_3$, where $\boldsymbol{\Pi}$ is a permutation matrix and $\boldsymbol{\Lambda}_i$ is a full rank diagonal matrix such that $\boldsymbol{\Lambda}_1\boldsymbol{\Lambda}_2\boldsymbol{\Lambda}_3 = \boldsymbol{I}$.

A tensor can be represented in a matrix form using the *matricization* operation. There are three common ways to matricize (or unfold) a third-order tensor, by stacking columns, rows, or fibers of the tensor to form a matrix. To be more precise let:

$$\underline{\boldsymbol{X}}(:, :, k) = \boldsymbol{X}_k \in \mathbb{R}^{I \times J}, \tag{2}$$

where $\boldsymbol{X}_k$ are the frontal slabs of tensor $\underline{\boldsymbol{X}}$ and in the context of this paper they model adjacency matrices, powers of the adjacency, node attributes or attribute similarity matrices. Then the mode-1, mode-2 and mode-3 unfoldings of $\underline{\boldsymbol{X}}$ can be cast as:

$$\boldsymbol{X}^{(1)} = \left[\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_K\right]^T = (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T \in \mathbb{R}^{JK \times I}, \tag{3}$$

$$\boldsymbol{X}^{(2)} = \left[\boldsymbol{X}_1^T, \boldsymbol{X}_2^T, \ldots, \boldsymbol{X}_K^T\right]^T = (\boldsymbol{C} \odot \boldsymbol{A})\boldsymbol{B}^T \in \mathbb{R}^{IK \times J}, \tag{4}$$

$$\boldsymbol{X}^{(3)} = \begin{bmatrix} \boldsymbol{X}_1(:, 1), \boldsymbol{X}_2(:, 1), \cdots, \boldsymbol{X}_K(:, 1) \\ \vdots \\ \boldsymbol{X}_1(:, J), \boldsymbol{X}_2(:, J), \cdots, \boldsymbol{X}_K(:, J) \end{bmatrix} = (\boldsymbol{B} \odot \boldsymbol{A})\boldsymbol{C}^T \in \mathbb{R}^{IJ \times K}. \tag{5}$$

3 PROBLEM STATEMENT

We begin the discussion with the definition of node embedding. Let $\mathcal{G} := \{\mathcal{V}, \mathcal{E}\}$ be a directed or undirected graph, with $\mathcal{V}$ being the set of $N = |\mathcal{V}|$ nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ being the set of edges. We are also given a set of attributes $\mathcal{A}$ for each node. Node embedding aims to map each node to a vector in $F-$dimensional Euclidean space. Formally, the node embedding task seeks for a function $f(\cdot) : \mathcal{G}, \mathcal{A} \rightarrow \mathbb{R}^{N \times F}$, where $F \ll N$. The node embeddings can be represented by matrix $\boldsymbol{E} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_N]^T$, where each row that contains the F-dimensional embedding of each node.

3.1 Related work

Recent work [2] proposed building a tensor $\underline{\boldsymbol{X}}$ whose first frontal slab $\boldsymbol{X}_1$ is the network adjacency matrix, while its second frontal slab $\boldsymbol{X}_2$ is the an attribute adjacency, obtained by computing the set of k nearest neighbors [28] of each node in attribute space. In other words, the attributes of a given node are viewed as a vector in Euclidean space, and the k closest attribute vectors of other nodes in the network are used to define the neighbors of the given node. The number of nearest neighbors is a parameter that needs to be tuned. A second adjacency matrix is produced this way, which however is not necessarily symmetric (even if the original network adjacency is). Joint analysis of these two adjacency matrices yields embeddings that reflect both pieces of information – but are not geometry-preserving, because (approximately) reproducing these adjacency matrices has no geometric motivation / interpretation.

In this work we propose a principled formulation that directly aims to produce an embedding that can reproduce the distances between nodes in terms of their network adjacency and in terms of their attributes. We find common latent dimensions that explain both sets of distances. With proper weighting, the resulting embedding vectors reproduce the adjacency distances; with another weighting, they reproduce the attribute distances (and these weights are a by-product of our analysis). Either way, the latent dimensions are derived from (and reflect) both sets of distances. This is why we call the approach *geometry preserving*. Depending on the downstream task, different weighting schemes would be more appropriate. Our formulation draws from multi dimensional scaling (MDS), which is briefly reviewed next.

3.2 Multi dimensional scaling

MDS is a distance-preserving mapping, visualization, and embedding tool [9, 24, 31, 35]. Given an $N \times N$ matrix $\boldsymbol{D}$ of distances between N entities, MDS seeks to find N points in low-dimensional space (typically 2- or 3-dimensional, for visualization purposes) that approximately exhibit the given distances. Various distances (and pseudo-distances) can be used for MDS. The most popular is the Euclidean distance, leading to the classical MDS, but there exist non-metric versions of MDS which seek to preserve ordering as opposed to distances [24]. We next briefly review classical MDS. Let $\boldsymbol{D}^{(2)} \in \mathbb{R}^{N \times N}$ be the matrix of squared distances between N entities, with $\boldsymbol{D}^{(2)}(i, j)$ being the squared distance between entity i and entity j. Now let $\boldsymbol{e}_i$ be the vector representation of entity i in a low F-dimensional Euclidean space. Then it holds that:

$$\boldsymbol{D}^{(2)}(i, j) = \|\boldsymbol{e}_i - \boldsymbol{e}_j\|^2 = \|\boldsymbol{e}_i\|^2 + \|\boldsymbol{e}_j\|^2 - 2\boldsymbol{e}_i^T\boldsymbol{e}_j \tag{6}$$

Since the objective is to learn the $\{\boldsymbol{e}_i\}_{i=1}^N$ we would like to end up with an expression that ignores the squared norms $\|\boldsymbol{e}_i\|^2, \|\boldsymbol{e}_j\|^2$ and will be easy to factor. In this direction we observe that:

$$\boldsymbol{D}^{(2)} = \boldsymbol{g}\boldsymbol{1}^T + \boldsymbol{1}\boldsymbol{g}^T - 2\boldsymbol{E}\boldsymbol{E}^T, \tag{7}$$

where $\boldsymbol{g} = \left[\boldsymbol{e}_1^T\boldsymbol{e}_1, \ldots, \boldsymbol{e}_N^T\boldsymbol{e}_N\right]^T$. Double centering both sides yields:

$$\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{D}^{(2)}\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right) = \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{g}\boldsymbol{1}^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right) + \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{1}\boldsymbol{g}^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right) - \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)2\boldsymbol{E}\boldsymbol{E}^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right), \tag{8}$$

which is equivalent to

$$-\frac{1}{2}\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{D}^{(2)}\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right) = \boldsymbol{E}\boldsymbol{E}^T, \tag{9}$$

since $\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{1} = 0$ and we can assume without loss of generality that matrix $\boldsymbol{E}$ is already centered. The solution for $\boldsymbol{E}$ is given

by

$$\boldsymbol{E} = \boldsymbol{U}\sqrt{\boldsymbol{\Lambda}_F}, \tag{10}$$

where $\boldsymbol{U} \in \mathbb{R}^{N\times F}$ is the matrix of F principal eigenvectors and $\boldsymbol{\Lambda}_F \in \mathbb{R}^{F\times F}$ a diagonal matrix with the F principal eigenvalues of $-\frac{1}{2}\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{D}^{(2)}\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)$. In the non-ideal case where $\boldsymbol{D}^{(2)}$ is inexact, assuming that the left hand side of (9) remains (or is projected to be) positive semidefinite, (10) gives the best vector representation of the entities in an F-dimensional space *after double-centering*, albeit that is not optimal from the viewpoint of preserving the original distances. For the latter, we need to resort to iterative algorithms that minimize a suitable cost (or *stress*) function, but that is often not necessary in practice.

MDS has also been generalized to the case where more than one distance matrices are available for a set of entities [7]. For example, the entities could be a set of N different products and K individuals are asked to rate their similarity or dissimilarity. This results in K different $N \times N$ distance matrices for the N products. To be more precise let $\boldsymbol{D}_k^{(2)} \in \mathbb{R}^{N\times N}$ be the k-th given distance matrix. Three-way MDS forms a third-order tensor $\underline{\boldsymbol{X}} \in \mathbb{R}^{N\times N\times K}$ as:

$$\underline{\boldsymbol{X}}(:,:,k) = \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{D}_k^{(2)}\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right) \tag{11}$$

and performs CPD of $\underline{\boldsymbol{X}}$ to find a joint F-dimensional representation of the entities.

3.3 GAGE: Geometry preserving Attributed Graph Embeddings

In the previous section we introduced the task of unsupervised node embedding. The objective of this task is to map each node of the network to a low dimensional vector representation in the Euclidean space. It is desirable that the low dimensional embedding contains as much connectivity and attribute information as possible and progress in this direction is the key to successful embeddings.

In this section, motivated by the benefits of MDS, we propose a novel unsupervised node embedding scheme that works with attributed networks. The proposed node embedding scheme attempts to preserve the network geometry inferred both from connectivity and attribute information. Furthermore, the node embeddings are unique. Note that uniqueness is a fundamental property that each embedding should enjoy. It offers a unique representation of each node, which is necessary for any form of interpretability and also guarantees that the embedding is permutation invariant. In other words any permuted version of the adjacency yields the same embedding for each node. Finally the proposed representation model is flexible in the sense that it can handle both directed and undirected graphs and does not require connectivity and attributed information for every node. In other words embeddings can be produced for nodes with either missing connectivity information or missing attributes.

Traditional MDS starts from a distance matrix and looks for vector representations of the nodes. In our setting, we are given the adjacency representation of each node along with a vector of attributes. The obvious approach would be to try and learn low-dimensional node embeddings directly from the high-dimensional graph and attribute representation of each node. However, since our objective is the produced embeddings to preserve the network geometry in terms of Euclidean distances, we propose to follow a different route. In particular, given the adjacency and the attributes of the network we compute distance matrices, one for the connectivity information and another for the attribute information. This transformation from adjacency and attributes to connectivity and attribute distances is the key to our proposed geometry preserving embeddings. Then we decompose the tensor of distances, using the CPD model, and produce the low-dimensional embeddings. As we see later in the section, the produced embeddings, which are formed from the CPD factors, can reproduce both the connectivity and attribute distances. Note that, from a computational viewpoint, instantiating the Euclidean distance matrices of connectivity and attributes might be prohibitive, since it destroys the sparsity structure. Interestingly, there is a elegant way to overcome it.

In order to facilitate the analysis let $\boldsymbol{S}_{\mathcal{G}} \in \{0,1\}^{N\times N}$ denote the adjacency matrix of graph $\mathcal{G}$ and $\mathcal{A} \in \mathbb{R}^{N\times d}$ be the matrix the contains in row i d attributes or features of vertex i. Also let $\boldsymbol{Y}_1 = \boldsymbol{S}_{\mathcal{G}}$ and $\boldsymbol{Y}_2 = \mathcal{A}$. Taking a closer look at equation (8) we observe that double centering the matirx of Euclidean distances between the rows of $\boldsymbol{Y}_1$ or $\boldsymbol{Y}_2$ is equal to double centering $\boldsymbol{Y}_1\boldsymbol{Y}_1^T$ or $\boldsymbol{Y}_2\boldsymbol{Y}_2^T$. This is due to the fact that $\boldsymbol{Y}_1$ or $\boldsymbol{Y}_2$ contain the generating vectors of the distance matrices and equation (7) always holds. We now transform the adjacency and attribute to distance matrices:

$$\boldsymbol{X}_1 = \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{Y}_1\boldsymbol{Y}_1^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right), \tag{12}$$

$$\boldsymbol{X}_2 = \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{Y}_2\boldsymbol{Y}_2^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right). \tag{13}$$

Note that $\boldsymbol{X}_1(i,j)$ denotes the squared Euclidean distance (after double centering) between $\boldsymbol{S}_G(i,:)$ and $\boldsymbol{S}_G(j,:)$, i.e., two rows of the adjacency matrix. Also $\boldsymbol{X}_2(i,j)$ denotes the squared Euclidean distance (after double centering) between $\mathcal{A}(i,:)$ and $\mathcal{A}(j,:)$, i.e., two attributed information of different nodes. It is important to mention that in most applications $\boldsymbol{S}_{\mathcal{G}}$ and $\mathcal{A}$ are sparse matrices which facilitates storage and computation requirements. Double centering these matrices automatically yields dense matrices. However, as we will see next our approach doesn't instantiate the dense $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ but works with sparse $\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2$, which is crucial to keep the computational and memory complexity of the algorithm low.

To compute the node embeddings of the attributed network, we propose to employ the following optimization scheme:

$$\min_{\boldsymbol{U},\boldsymbol{\Lambda}_1,\boldsymbol{\Lambda}_2} \|\boldsymbol{X}_1 - \boldsymbol{U}\boldsymbol{\Lambda}_1\boldsymbol{U}^T\|_F^2 + \|\boldsymbol{X}_2 - \boldsymbol{U}\boldsymbol{\Lambda}_2\boldsymbol{U}^T\|_F^2, \tag{14}$$

where $\boldsymbol{U} \in \mathbb{R}^{N\times F}$ and $\boldsymbol{\Lambda}_1, \boldsymbol{\Lambda}_2$ are real and positive valued $F \times F$ diagonal matrices. Problem (14) is the rank F CPD of tensor $\underline{\boldsymbol{X}} \in \mathbb{R}^{N\times N\times 2}$, with frontal slabs $\underline{\boldsymbol{X}}(:,:,1) = \boldsymbol{X}_1$ and $\underline{\boldsymbol{X}}(:,:,2) = \boldsymbol{X}_2$. The CPD model for $\underline{\boldsymbol{X}}$ takes the form:

$$\underline{\boldsymbol{X}} = [\![\boldsymbol{U},\boldsymbol{U},\boldsymbol{C}]\!],\ \boldsymbol{C}(i,:)^T = \mathrm{diag}\left(\boldsymbol{\Lambda}_i\right),\ i = 1,2, \tag{15}$$

where $\mathrm{diag}\left(\boldsymbol{\Lambda}_i\right)$ is the diagonal vector of $\boldsymbol{\Lambda}_i$. The proposed embedding for vertex i is :

$$\boldsymbol{e}_i = \boldsymbol{E}(i,:)^T = \mathrm{diag}\left(\sqrt{\lambda\boldsymbol{C}(:,1)^T + (1-\lambda)\boldsymbol{C}(:,2)^T}\right)\boldsymbol{U}(i,:)^T, \tag{16}$$

where $\text{diag}\left(\sqrt{\lambda C(:,1)^T + (1-\lambda)C(:,2)^T}\right)$ gives the diagonal matrix of vector $\sqrt{\lambda C(:,1)^T + (1-\lambda)C(:,2)^T}$. Note that the $0 \leq \lambda \leq 1$ parameter balances the contribution of each distance measure (connectivity or attribute) in the final embedding. For $\lambda = 1$ the focus is completely on the connectivity distances, whereas for $\lambda = 0$ the emphasis is on the attribute distances.

Invoking the uniqueness properties o the CPD (see Theorem 1 for details) we have shown the following result:

Result 1. *If tensor $\underline{\boldsymbol{X}}$ has indeed low-rank, F, there exist vectors in F dimensional space that generate the given sets of distances (with appropriate weights). Then the* GAGE *embeddings for the correct F are unique, permutation invariant and will exactly reproduce both sets of distances for $\lambda = 0$ and $\lambda = 1$.*

The above result also implies that embeddings of dimension less than F cannot reconstruct the set of distances and embeddings of dimension larger than F are not unique.

4 ALGORITHMIC FRAMEWORK

In this section we discuss the algorithmic aspects of our approach.

4.1 The GAGE algorithm

The computation of the proposed node embeddings boils down to solving the problem in (14). This is a CPD problem of an $N \times N \times 2$ tensor with a special sparsity structure on the frontal slabs. CPD computation is a non-convex optimization problem and in general NP-hard. However, exact CPD can be reduced to eigenvalue decomposition (EVD) in certain cases – notably when tensor rank is low enough [11, 30]. Such an approach is not guaranteed to produce the optimal solution, but it often works well in practice, and it also serves as good initialization for more sophisticated optimization approaches. Developing a computationally efficient algebraic initialization approach to tackle the problem in (14) is therefore an important pivot for the proposed algorithm. This is GAGE-EVD, which is summarized in Algorithm 1. The first step of the approach is to form the doubly centered frontal slabs. Note that instantiating $\boldsymbol{X}_1,\ \boldsymbol{X}_2$ is not required and we can directly work with $\boldsymbol{Y}_1,\ \boldsymbol{Y}_2$ as shown in Appendix A.1; GAGE-EVD exploits sparsity and the special problem structure to mitigate memory and complexity requirements. The next step is to compute the F principal eigenvectors $\boldsymbol{V}$ of $\boldsymbol{X}_1^T\boldsymbol{X}_1 + \boldsymbol{X}_2^T\boldsymbol{X}_2$. Towards this, end we employ the orthogonal iterations method [15] which also exploits the special sparsity structure to enable lightweight computations. Finally, we form $\boldsymbol{S}_1 = \boldsymbol{V}^T\boldsymbol{X}_1\boldsymbol{V},\ \boldsymbol{S}_2 = \boldsymbol{V}^T\boldsymbol{X}_2\boldsymbol{V}$ which are dense but small ($F \times F$) matrices and compute the eigenvalue decomposition of $\boldsymbol{S}_2\boldsymbol{S}_1^{-1}$. Then $\boldsymbol{U}$ is computed as $\boldsymbol{U}^T = \tilde{\boldsymbol{U}}^{-1}\boldsymbol{S}_1$. In terms of computational complexity, the main bottleneck of GAGE-EVD is computing the EVD of $\boldsymbol{X}_1^T\boldsymbol{X}_1 + \boldsymbol{X}_2^T\boldsymbol{X}_2$. Using the orthogonal iterations method, this EVD can be computed efficiently in $O(NF^2)$ flops. The remaining operations involve $F \times F$ matrices and are computationally light. Detailed description of the algorithmic updates along with computational complexity and memory requirements is given in Appendix A.1.

After computing an initial estimate of matrix $\boldsymbol{U}$, we feed it to the main GAGE algorithm, which is summarized in Algorithm 2. To tackle the problem in (14) GAGE follows an alternating least squares approach, with the first two factors $\boldsymbol{U}, \boldsymbol{U}'$ not constrained to be equal (see Algorithm 2). In each update, we fix two factors and solve for the remaining one. We repeat this procedure in an alternating fashion. The update for each step is a linear system of equations and can be solved efficiently without instantiating the dense tensor $\underline{\boldsymbol{X}}$ or any of the Khatri-Rao products, i.e., $(\boldsymbol{C} \odot \boldsymbol{U}')$, $(\boldsymbol{C} \odot \boldsymbol{U}')$, $(\boldsymbol{U}' \odot \boldsymbol{U})$. The details are presented in Appendix A.2. Note that due to the algebraic initialization, the GAGE algorithm converges in only a few steps (usually fewer than 10) in our experiments.

Algorithm 1: GAGE-EVD

Input: $\boldsymbol{Y}_1 = \boldsymbol{S}_{\mathcal{G}}, \boldsymbol{Y}_2 = \boldsymbol{\mathcal{A}},\ F$.
Output: $\boldsymbol{U}$.
$\boldsymbol{X}_1 = \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{Y}_1\boldsymbol{Y}_1^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)$;
$\boldsymbol{X}_2 = \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{Y}_2\boldsymbol{Y}_2^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)$;
$\boldsymbol{V}\Sigma\boldsymbol{V}^T \leftarrow \text{EVD}(\boldsymbol{X}_1^T\boldsymbol{X}_1 + \boldsymbol{X}_2^T\boldsymbol{X}_2, F)$;
$\boldsymbol{S}_1 = \boldsymbol{V}^T\boldsymbol{X}_1\boldsymbol{V},\ \boldsymbol{S}_2 = \boldsymbol{V}^T\boldsymbol{X}_2\boldsymbol{V}$;
$\tilde{\boldsymbol{U}} \leftarrow \text{EVD}\left(\boldsymbol{S}_2\boldsymbol{S}_1^{-1}\right)$;
$\boldsymbol{U}^T = \tilde{\boldsymbol{U}}^{-1}\boldsymbol{V}^T$;

Algorithm 2: GAGE

Input: $\boldsymbol{Y}_1 = \boldsymbol{S}_{\mathcal{G}}, \boldsymbol{Y}_2 = \boldsymbol{\mathcal{A}},\ \boldsymbol{U}$.
Output: $\boldsymbol{E}$.
$\boldsymbol{X}_1 = \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{Y}_1\boldsymbol{Y}_1^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)$;
$\boldsymbol{X}_2 = \left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)\boldsymbol{Y}_2\boldsymbol{Y}_2^T\left(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\right)$;
$\underline{\boldsymbol{X}}(:,:,1) = \boldsymbol{X}_1,\ \underline{\boldsymbol{X}}(:,:,2) = \boldsymbol{X}_2$;
$\boldsymbol{C} \leftarrow \text{solve } \boldsymbol{X}^{(3)} = (\boldsymbol{U} \odot \boldsymbol{U})\boldsymbol{C}^T$;
$\boldsymbol{U}' = \boldsymbol{U}$;
repeat
  $\boldsymbol{U} \leftarrow \text{solve } \boldsymbol{X}^{(1)} = (\boldsymbol{C} \odot \boldsymbol{U}')\boldsymbol{U}^T$;
  $\boldsymbol{U}' \leftarrow \text{solve } \boldsymbol{X}^{(2)} = (\boldsymbol{C} \odot \boldsymbol{U})\boldsymbol{U}'^T$;
  $\boldsymbol{C} \leftarrow \text{solve } \boldsymbol{X}^{(3)} = (\boldsymbol{U}' \odot \boldsymbol{U})\boldsymbol{C}^T$;
until convergence
$\boldsymbol{E} = \boldsymbol{U}\,\text{diag}\left(\sqrt{\lambda C(:,1)^T + (1-\lambda)C(:,2)^T}\right)$;

5 EXPERIMENTS

In this section we demonstrate the performance of the proposed algorithmic framework and showcase its effectiveness in experiments with real attributed network data. All algorithms were implemented in Matlab or Python, and executed on a Linux server comprising 8 cores at 3.6GHz with 32GB RAM.

5.1 Data

We used the following real-world networks (see also Table 2).

- **BlogCatalog**. A social network of bloggers in BlogCatalog platform. Each blogger uses several keywords to describe their blogs. These keywords have been used as attributes for the node-bloggers. There are 6 different classes of bloggers according to the content of their blogs. The attributes dimension represents the dictionary and each node is encoded with a sparse bag-of-words representation.
- **WebKB** [14]. A network of webpages from computer science departments categorized into 5 topics: faculty, student, project, course, other. The attributes dimension is a dictionary of words that appear in the webpages.
- **Wikipedia** [39]. A network of documents and their Wikipedia links. The documents are grouped into 19 classes and the attribute information corresponds to sparse TFIDF features.

Table 2: Datasets

Dataset	# Vertices	# Edges	Attribute dimension	# Classes	Network Type	Feature Type
Wikipedia	2,405	23,192	4,973	19	Language	Text associated info
WebKB	877	2,776	1,703	5	Citation	Unique words
BlogCatalog	5,196	686,972	8,189	6	Social	Keywords

5.2 Baselines

- **Deepwalk** [27]. Deepwalk generates truncated random walks from each node, to learn low dimensional representations of nodes using a SkiGram model. We set the number of walks per node $\gamma = 80$, walk length $t = 40$ and window size $w = 10$ as suggested in [27]. This method does not use the attributes, and it is not expected to work as well as the other methods that do. We include it, since it remains a strong contender, and as a means to gauge the improvement afforded by having access to the node attributes.
- **T-Pine** [2]. A tensor factorization based approach. The first frontal slab is the adjacency of the graph and the second frontal slab is the a k nearest neighbor matrix computed using the distances between the node attributes. The k nearest neighbor parameter is set to $k = 8$ for Wikipedia, $k = 40$ for WebKB as suggested in [2] and $k = 50$ for BlogCatalog.
- **Graph-AE** [22]. A graph convolutional network (GCN) generalization for unsupervised node embedding. Graph-AE uses a (GCN) encoder and a simple inner product decoder.
- **Graph-VAE** [22]. A variational auto encoder (VAE) alternative to Graph-AE. Both Graph-AE and Graph-VAE are trained using 200 epochs and 0.01 learning rate. The dimension of the hidden layer is twice the number of the embedding dimension. We use 5% of the data for validation.
- **TADW** [39]. Text associated Deepwalk (TADW) employs a matrix factorization framework to learn network representations using the adjacency matrix as well as textual information features.
- **DGI** [37]. Deep Graph Infomax (DGI) uses a graph convolutional neural network architecture to learn node embeddings for attributed networks in an unsupervised manner. We train for maximum 1000 epochs using the code provided by the authors and set the 'patience' parameter equal to 20 and learning rate equal to 0.001, as suggested in [37].
- **AGE** [10]. Adaptive Graph Encoder for Attributed Graph Embedding (AGE) uses a Laplacian smoothing filter along with an adaptive encoder to perform attributed node embedding. We use 400 epochs and learning rate equal to 10^{-3} for training, as suggested in the author's code.
- **DANE** [13]. Deep Attributed Network Embedding (DANE) adopts a 2-branch encoder-decoder architecture to learn attributed node embeddings. The first branch is associated with the connectivity information of the network, whereas the second one utilizes the attribute information. We use 500 epochs to train the autoencoder with learning rate and dropout probability equal to 10^{-5} and 0.2 respectively.

For all baselines we use the publicly available code provided by the authors.

5.3 Node classification

We first test the performance of the proposed GAGE along with the baselines in a node classification task. The procedure is divided in two steps. In the first step the algorithms learn the node embeddings in a unsupervised manner, i.e., without using label information. In the second step the labels along with the learned embeddings are split into training and testing sets. Then the training data are fed to a one-versus-all logistic regression classifier with l_2 norm regularization. We test 3 different training-testing splits, i.e., 0.9-0.1, 0.5-0.5, and 0.1-0.9 and run 10 shuffles for each split. To assess the performance of the competing algorithms we measure the average micro and macro F1 score for 2 different embedding dimensions. For the GAGE embeddings we set $\lambda = 0.8$. The results for the three different datasets are presented in Tables 3, 4, 5.

It is clear from the tables that the proposed GAGE significantly outperforms the baselines in both micro and macro F1 score, where T-PINE usually comes second. In the Wikipedia dataset there are instances where T-PINE is slightly better in micro F1 but GAGE is better in macro F1. Taking into consideration that the Wikipedia dataset consists of 19 classes and some classes are skewed, macro F1 score is far more significant in this dataset. Note that Graph-AE and Graph-VAE show in general very weak classification performance and especially for the $F = 256$ in BlogCatalog they fail to produce acceptable results. We also notice that some baselines, that take attributes into account, produce weaker results compared to Deepwalk that only uses connectivity information. This is due to the fact that the considered datasets have missing attributes and certain baselines failed to be efficient under this challenging setting.

5.4 Link prediction

The proposed embeddings are also tested for link prediction – see Appendix A. We observed that GAGE achieves high prediction performance, but is sometimes outperformed by graph encoders and auto-encoders as [10, 13, 21]. The reason is that graph encoders and auto-encoders treat the unobserved links as unknown rather than non-existing. This benefits link prediction but on the downside renders these approaches task-specific and results in weak classification performance. On the contrary, GAGE is a global approach, offers elite performance in both tasks and overall produces more informative node embeddings.

5.5 Sensitivity analysis and running time

We examined the effect of parameter λ in the performance of GAGE for node classification and link prediction. The details are relegated to Appendix B due to space limitations. In a nutshell, we observe that classification performance is consistent for $\lambda \in [0.1, 0.9]$ and the best performance is usually achieved for $\lambda \in [0.5, 0.9]$. Regarding link prediction, the best results are achieved when $\lambda = 1$ and the performance deteriorates as lambda decreases.

Table 3: Average score and standard deviation over 10 shuffles for `Wikipedia`

Algorithm	dimension	micro (0.9)	macro (0.9)	micro (0.5)	macro (0.5)	micro (0.1)	macro (0.1)
GAGE	64	**0.7402 ± 0.0308**	**0.5331 ± 0.0239**	**0.7303 ± 0.0125**	**0.5262 ± 0.0217**	**0.6309 ± 0.0217**	**0.423 ± 0.0246**
	128	0.7656 ± 0.0255	**0.5924 ± 0.0337**	0.736 ± 0.0104	**0.5802 ± 0.0198**	**0.649 ± 0.0179**	**0.4728 ± 0.0261**
T-PINE	64	0.6788 ± 0.0312	0.4039 ± 0.0158	0.6619 ± 0.0072	0.3949 ± 0.0047	0.5912 ± 0.0139	0.3535 ± 0.0042
	128	**0.766 ± 0.0234**	0.523 ± 0.0183	**0.745 ± 0.009**	0.5069 ± 0.0121	0.6364 ± 0.0081	0.4205 ± 0.0144
Deepwalk	64	0.6177 ± 0.0309	0.3632 ± 0.0213	0.6136 ± 0.0038	0.3736 ± 0.0119	0.5773 ± 0.0084	0.3415 ± 0.0126
	128	0.6236 ± 0.0333	0.362 ± 0.0175	0.614 ± 0.006	0.3731 ± 0.0111	0.5811 ± 0.0095	0.3444 ± 0.0126
Graph-AE	64	0.6759 ± 0.0314	0.4512 ± 0.0335	0.6481 ± 0.0117	0.4254 ± 0.0193	0.5669 ± 0.0075	0.3452 ± 0.0121
	128	0.6747 ± 0.0372	0.4327 ± 0.0346	0.6584 ± 0.0082	0.4287 ± 0.0203	0.5773 ± 0.01	0.3536 ± 0.0115
Graph-VAE	64	0.6283 ± 0.03	0.4108 ± 0.0297	0.6069 ± 0.009	0.3804 ± 0.0137	0.5592 ± 0.0112	0.3323 ± 0.0124
	128	0.67 ± 0.0394	0.436 ± 0.028	0.6404 ± 0.0119	0.4098 ± 0.0195	0.5742 ± 0.0107	0.3431 ± 0.0109
TADW	64	0.7008 ± 0.0258	0.4541 ± 0.0244	0.6990 ± 0.0142	0.4781 ± 0.0156	0.6160 ± 0.0121	0.3996 ± 0.0126
	128	0.7510 ± 0.0345	0.5378 ± 0.0373	0.7168 ± 0.0135	0.5170 ± 0.0224	0.6309 ± 0.0159	0.4221 ± 0.0218
DGI	64	0.5523 ± 0.0258	0.2806 ± 0.0095	0.4927 ± 0.0182	0.2271 ± 0.0124	0.3157 ± 0.0323	0.0741 ± 0.0139
	128	0.5299 ± 0.0254	0.252 ± 0.0139	0.4563 ± 0.0171	0.1825 ± 0.0119	0.2924 ± 0.0458	0.066 ± 0.0148
AGE	64	0.6996 ± 0.0228	0.4398 ± 0.009	0.6826 ± 0.0138	0.4261 ± 0.0124	0.6038 ± 0.0217	0.3448 ± 0.0258
	128	0.7058 ± 0.0323	0.452 ± 0.0182	0.6909 ± 0.0094	0.437 ± 0.0084	0.5833 ± 0.0217	0.3105 ± 0.0254
DANE	64	0.5029 ± 0.0383	0.2575 ± 0.0314	0.4259 ± 0.0243	0.1926 ± 0.0134	0.2409 ± 0.025	0.0575 ± 0.0124
	128	0.6734 ± 0.0339	0.4141 ± 0.022	0.6565 ± 0.0112	0.3976 ± 0.0093	0.5321 ± 0.0208	0.2706 ± 0.0219

Table 4: Average score over 10 shuffles for `WebKB`

Algorithm	dimension	micro (0.9)	macro (0.9)	micro (0.5)	macro (0.5)	micro (0.1)	macro (0.1)
GAGE	64	**0.8852 ± 0.0375**	**0.7588 ± 0.0506**	**0.8547 ± 0.0221**	**0.7005 ± 0.0228**	**0.7722 ± 0.0233**	**0.5701 ± 0.0352**
	128	**0.8864 ± 0.037**	**0.7618 ± 0.0645**	**0.8601 ± 0.0148**	**0.7024 ± 0.0264**	**0.7566 ± 0.0256**	**0.5419 ± 0.0372**
T-PINE	64	0.8148 ± 0.0318	0.6504 ± 0.0633	0.8016 ± 0.0192	0.6361 ± 0.0224	0.7033 ± 0.018	0.5204 ± 0.0185
	128	0.7989 ± 0.0297	0.6394 ± 0.0632	0.7743 ± 0.0144	0.6141 ± 0.0241	0.681 ± 0.0201	0.4822 ± 0.0166
Deepwalk	64	0.5081 ± 0.0543	0.2627 ± 0.0284	0.4786 ± 0.0206	0.2448 ± 0.0217	0.4367 ± 0.0152	0.2228 ± 0.0175
	128	0.4977 ± 0.049	0.2914 ± 0.0437	0.4674 ± 0.0207	0.2487 ± 0.0242	0.4447 ± 0.015	0.2249 ± 0.0177
Graph-AE	64	0.4591 ± 0.0306	0.1261 ± 0.0061	0.4722 ± 0.0144	0.1294 ± 0.0029	0.4767 ± 0.0079	0.1373 ± 0.0119
	128	0.4591 ± 0.0306	0.1257 ± 0.0058	0.4715 ± 0.0146	0.1281 ± 0.0027	0.4732 ± 0.0056	0.1285 ± 0.001
Graph-VAE	64	0.5261 ± 0.0322	0.2435 ± 0.0275	0.5276 ± 0.0135	0.2502 ± 0.0123	0.4985 ± 0.0165	0.2483 ± 0.0267
	128	0.542 ± 0.0446	0.2489 ± 0.0206	0.5376 ± 0.0207	0.2521 ± 0.012	0.5009 ± 0.0185	0.2434 ± 0.0258
TADW	64	0.6931 ± 0.0344	0.5368 ± 0.0562	0.6646 ± 0.0226	0.4887 ± 0.0355	0.5988 ± 0.0190	0.3889 ± 0.0250
	128	0.7511 ± 0.0404	0.6176 ± 0.0849	0.7200 ± 0.0252	0.5539 ± 0.0305	0.6287 ± 0.0213	0.4197 ± 0.0306
DGI	64	0.4705 ± 0.0378	0.147 ± 0.0199	0.4797 ± 0.0163	0.1444 ± 0.0067	0.4762 ± 0.0068	0.1336 ± 0.0036
	128	0.4727 ± 0.0374	0.1472 ± 0.0205	0.4772 ± 0.0146	0.1378 ± 0.0047	0.4746 ± 0.0069	0.1308 ± 0.0037
AGE	64	0.5205 ± 0.0312	0.2225 ± 0.026	0.5041 ± 0.02	0.1955 ± 0.0155	0.4618 ± 0.0196	0.164 ± 0.0283
	128	0.517 ± 0.0334	0.2133 ± 0.0208	0.5107 ± 0.0183	0.2003 ± 0.0066	0.4654 ± 0.0262	0.1663 ± 0.0335
DANE	64	0.6136 ± 0.0356	0.2854 ± 0.0143	0.5503 ± 0.025	0.2324 ± 0.0161	0.4903 ± 0.0462	0.1776 ± 0.04
	128	0.7295 ± 0.0457	0.4309 ± 0.0378	0.7064 ± 0.0219	0.4089 ± 0.026	0.6287 ± 0.0292	0.3181 ± 0.0314

Table 5: Average score and standard deviation over 10 shuffles for BlogCatalog

Algorithm	dimension	micro (0.9)	macro (0.9)	micro (0.5)	macro (0.5)	micro (0.1)	macro (0.1)
GAGE	128	0.9233 ± 0.009	0.9208 ± 0.0095	**0.9191 ± 0.0028**	**0.9171 ± 0.0027**	**0.8858 ± 0.0101**	**0.8842 ± 0.0098**
	256	**0.9538 ± 0.0082**	**0.9527 ± 0.0082**	**0.9457 ± 0.0017**	**0.9447 ± 0.0018**	**0.912 ± 0.0066**	**0.9109 ± 0.0066**
T-PINE	128	**0.9281 ± 0.0087**	**0.9263 ± 0.0093**	0.9145 ± 0.0045	0.913 ± 0.0047	0.8577 ± 0.0055	0.8563 ± 0.0053
	256	0.9213 ± 0.0098	0.9196 ± 0.0097	0.9076 ± 0.0043	0.9061 ± 0.0044	0.8681 ± 0.0048	0.867 ± 0.0048
Deepwalk	128	0.6937 ± 0.0212	0.6802 ± 0.0218	0.681 ± 0.0056	0.673 ± 0.0059	0.6187 ± 0.0083	0.6117 ± 0.0081
	256	0.6923 ± 0.0197	0.6796 ± 0.0207	0.6823 ± 0.0051	0.6743 ± 0.0054	0.619 ± 0.0089	0.6121 ± 0.0086
Graph-AE	128	0.2521 ± 0.0128	0.179 ± 0.0077	0.2455 ± 0.0086	0.1806 ± 0.0133	0.2547 ± 0.0107	0.1454 ± 0.0154
	256	–	–	–	–	–	–
Graph-VAE	128	0.5306 ± 0.01	0.4896 ± 0.0119	0.5182 ± 0.0092	0.4754 ± 0.0128	0.467 ± 0.0149	0.4204 ± 0.021
	256	–	–	–	–	–	–
TADW	128	0.8504 ± 0.0106	0.8483 ± 0.012	0.8464 ± 0.0043	0.8442 ± 0.0044	0.8296 ± 0.0046	0.8284 ± 0.0042
	256	0.8485 ± 0.0102	0.8466 ± 0.0118	0.8446 ± 0.0041	0.8424 ± 0.0042	0.829 ± 0.0048	0.8278 ± 0.0042
DGI	128	0.6017 ± 0.0136	0.5624 ± 0.0136	0.5786 ± 0.0149	0.527 ± 0.0208	0.3693 ± 0.0583	0.2761 ± 0.0602
	256	0.6163 ± 0.0175	0.5642 ± 0.0188	0.595 ± 0.0157	0.5376 ± 0.0179	0.3236 ± 0.0729	0.2235 ± 0.0683
AGE	128	0.7279 ± 0.0174	0.7219 ± 0.0183	0.7109 ± 0.0063	0.7053 ± 0.0067	0.7279 ± 0.0174	0.7219 ± 0.0183
	256	0.726 ± 0.0147	0.7195 ± 0.0154	0.7115 ± 0.0052	0.7057 ± 0.0055	0.6708 ± 0.0097	0.658 ± 0.0105
DANE	128	0.4821 ± 0.0126	0.4637 ± 0.0153	0.4654 ± 0.0118	0.4364 ± 0.0126	0.4821 ± 0.0126	0.4637 ± 0.0153
	256	0.6748 ± 0.0149	0.6597 ± 0.016	0.6488 ± 0.01	0.6207 ± 0.015	0.4206 ± 0.0546	0.3407 ± 0.0715

Table 6: Running time (sec)

Dataset	GAGE	T-PINE	Deepwalk	G-AE	G-VAE	TADW	DGI	AGE	DANE
Wiki	32.2	79.5	267	47.9	49.7	**8.1**	42.7	202.1	155+1597.2
WebKB	**2.2**	80.1	73.8	20	20	2.7	10.7	14.2	51.9+464.1
BCatalog	**17.7**	628	653.8	347.6	340.2	63.4	269	1458.1	416.1+4111.8

We also measured the running time required for our proposed GAGE and the baselines to produce 128-dimensional embeddings for the three datasets. The results are presented in Table 6. DANE requires additional time (indicated after the plus sign) to compute random walks. It is clear that the proposed GAGE is the fastest for WebKB and BlogCatalog, whereas TADW is the fastest for Wikipedia.

6 CONCLUSIONS

In this paper we proposed GAGE, a novel tensor-based approach for unsupervised node embedding of attributed networks. GAGE leverages the favorable properties of multi dimensional scaling and canonical polyadic decomposition and provides embeddings that preserve the geometry of both network connectivity and attributes. Although the proposed approach works with distance matrices rather than the original adjacency and attributes the algorithm can still exploit the sparsity structure of the graph and the attributes and admits a scalable and lightweight implementation. Experiments with real world benchmark networks showcase the effectiveness of the proposed GAGE on downstream tasks.

A APPENDIX: EFFICIENT CPD COMPUTATIONS FOR MDS TENSOR

We are given an adjacency matrix $Y_1 = S_{\mathcal{G}} \in \{0,1\}^{N\times N}$ and a matrix of node attributes $Y_2 = \mathcal{A} \in \mathbb{R}^{N\times d}$. We are interested in computing the CPD of tensor $\underline{X}$ with:

$$\underline{X}(:,:,1) = X_1 = \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_1 Y_1^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right), \quad (17)$$

$$\underline{X}(:,:,2) = X_2 = \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_2 Y_2^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) \quad (18)$$

The objective of this Appendix is to show how to perform this CPD computation by exploiting the special sparsity structure and without instantiating a dense $\underline{X}$.

A.1 GAGE-EVD

The first step of GAGE algorithm involves an eigenvalue decomposition. The bottleneck operation is:

$$V\Sigma V^T \leftarrow \text{EVD}\left(X^{(1)^T} X^{(1)}, F\right) \quad (19)$$

First let us observe the structure of matrix $X^{(1)^T} X^{(1)}$. Note that $X^{(1)} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$, and $X_1,\ X_2$ are both symmetric matrices.

$$X^{(1)^T} X^{(1)} = \begin{bmatrix} X_1^T X_2^T \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = X_1^T X_1 + X_2^T X_2 = \quad (20)$$

$$\left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_1 Y_1^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_1 Y_1^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) + \quad (21)$$

$$\left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_2 Y_2^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_2 Y_2^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right), \quad (22)$$

since $\left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right)\left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) = \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right)$. To compute the EVD of $X^{(1)^T} X^{(1)}$ we resort to the orthogonal iterations method [15]. The steps are summarized as follows:

- Initialize $Q_0 \in \mathbb{R}^{N\times F}$: orthogonal matrix
  repeat:
- $W_k = \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_1 Y_1^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_1 Y_1^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Q_{k-1} +$
  $\left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_2 Y_2^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_2 Y_2^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Q_{k-1}$
- $Q_k \leftarrow \text{QR}\ (W_k)$
  until convergence

It is clear that the above procedure does not instantiate $X_1,\ X_2$ and works directly with $Y_1,\ Y_2$. In the first step of the loop every computation is either a sparse or rank 1 multiplication which can be performed efficiently. The computationally more intensive computation lies in the QR computation of matrix W_k. The complexity of this step is $O(NF^2)$ which is linear in the number of nodes.

A.2 Sparsity aware GAGE

Now we study the ALS updates in GAGE algorithm. The update for U can be written as:

$$U \leftarrow \text{solve}\left(\left(C^T C) * (U^{'T} U'\right)\right) U^T = \left(C \odot U'\right)^T X^{(1)}. \quad (23)$$

The matrix-matrix multiplication in the right hand side exploits the special structure of $X^{(1)}$:

$$\left(C \odot U'\right)^T X^{(1)} = \begin{bmatrix} U' \text{diag}\,(C(1,:)) \\ U' \text{diag}\,(C(2,:)) \end{bmatrix}^T \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} =$$

$$\sum_{k=1}^{2} \text{diag}\,(C(k,:))\, U^{'T} \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_k Y_k^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right). \quad (24)$$

It follows that the number of flops required to compute (24) is $O(sF)$, where $s = s_1 + s_2$ and $s_1,\ s_2$ are the number of non-zeros in Y_1, Y_2 respectively. Furthermore, $X_1,\ X_2$ are not being instantiated. The same principles hold for the update of U':

$$U' \leftarrow \text{solve}\left(\left(C^T C) * (U^T U)\right)\right) U^{'T} = (C \odot U)^T X^{(2)}. \quad (25)$$

$$(C \odot U)^T X^{(2)} = \begin{bmatrix} U \text{diag}\,(C(1,:)) \\ U \text{diag}\,(C(2,:)) \end{bmatrix}^T \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} =$$

$$\sum_{k=1}^{2} \text{diag}\,(C(k,:))\, U^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right) Y_k Y_k^T \left(I - \frac{1}{N}\mathbf{1}\mathbf{1}^T\right). \quad (26)$$

The update of C can be written as:

$$C \leftarrow \text{solve}\left(\left(U^{'T} U') * (U^T U)\right)\right) C^T = \left(U' \odot U\right)^T X^{(3)}. \quad (27)$$

To avoid instantiating $U' \odot U$, we observe that:

$$\left(U' \odot U\right)^T X^{(3)} = \begin{bmatrix} U(:,1)^T X_1 U'(:,1), U(:,1)^T X_2 U'(:,1) \\ \vdots \\ U(:,F)^T X_1 U'(:,F), U(:,F)^T X_2 U'(:,F) \end{bmatrix} \quad (28)$$

The operation in (28) avoids storing $U' \odot U$. Furthermore, the formula in (12) is used for $X_1,\ X_2$, which exploits the structure of $Y_1,\ Y_2$ and does not instantiate $X_1,\ X_2$. The overall operation can be computed efficiently in $O(max\{NF, sF\})$ flops.

ACKNOWLEDGMENTS

Research was partially supported by NSF IIS-1908070 and ARO W911NF1910407.

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*. 37–48.

[2] Saba A Al-Sayouri, Ekta Gujral, Danai Koutra, Evangelos E Papalexakis, and Sarah S Lam. 2018. t-PNE: tensor-based predictable node embeddings. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 491–494.

[3] Albert-László Barabási et al. 2016. *Network science*. Cambridge university press.

[4] Dimitris Berberidis and Georgios B Giannakis. 2019. Node embedding with adaptive similarities for scalable learning over graphs. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 891–900.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations.. In *AAAI*, Vol. 16. 1145–1152.

[7] J Douglas Carroll and Jih-Jie Chang. 1970. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika* 35, 3 (1970), 283–319.

[8] Luca Chiantini and Giorgio Ottaviani. 2012. On generic identifiability of 3-tensors of small rank. *SIAM J. Matrix Anal. Appl.* 33, 3 (2012), 1018–1037.

[9] Michael AA Cox and Trevor F Cox. 2008. Multidimensional scaling. In *Handbook of data visualization*. Springer, 315–347.

[10] Ganqu Cui, Jie Zhou, Cheng Yang, and Zhiyuan Liu. 2020. Adaptive Graph Encoder for Attributed Graph Embedding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 976–985.

[11] Ignat Domanov and Lieven De Lathauwer. 2014. Canonical polyadic decomposition of third-order tensors: reduction to generalized eigenvalue decomposition. *SIAM J. Matrix Anal. Appl.* 35, 2 (2014), 636–660.

[12] David Easley, Jon Kleinberg, et al. 2010. *Networks, crowds, and markets*. Vol. 8. Cambridge university press Cambridge.

[13] Hongchang Gao and Heng Huang. 2018. Deep attributed network embedding. In *Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*.

[14] Lise Getoor. 2005. Link-based classification. In *Advanced methods for knowledge discovery from complex data*. Springer, 189–207.

[15] GH Golub and CF Van Loan. 2013. Matrix Computations 4th Edition The Johns Hopkins University Press. *Baltimore, MD* (2013).

[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[18] Richard A Harshman, Margaret E Lundy, et al. 1994. PARAFAC: Parallel factor analysis. *Computational Statistics and Data Analysis* 18, 1 (1994), 39–72.

[19] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. 731–739.

[20] Charilaos I Kanatsoulis and Nicholas D Sidiropoulos. 2021. TeX-Graph: Coupled tensor-matrix knowledge-graph embedding for COVID-19 drug repurposing. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 603–611.

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[22] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[23] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.

[24] Joseph B Kruskal. 1978. *Multidimensional scaling*. Number 11. Sage.

[25] Mark Newman. 2018. *Networks*. Oxford university press.

[26] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1105–1114.

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[28] Leif E Peterson. 2009. K-nearest neighbor. *Scholarpedia* 4, 2 (2009), 1883.

[29] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 459–467.

[30] Eugenio Sanchez and Bruce R Kowalski. 1990. Tensorial resolution: a direct trilinear decomposition. *Journal of Chemometrics* 4, 1 (1990), 29–45.

[31] Susan S Schiffman, M Lance Reynolds, and Forrest W Young. 1981. *Introduction to multidimensional scaling*. Academic press New York.

[32] Blake Shaw and Tony Jebara. 2009. Structure preserving embedding. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 937–944.

[33] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. 2017. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing* 65, 13 (2017), 3551–3582.

[34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[35] Warren S Torgerson. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* 17, 4 (1952), 401–419.

[36] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*. 539–548.

[37] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. *ICLR (Poster)* 2, 3 (2019), 4.

[38] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1225–1234.

[39] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network representation learning with rich text information. In *Proceedings of the 24th International Conference on Artificial Intelligence*. 2111–2117.

A LINK PREDICTION

We test the performance of the competing algorithms in the link prediction task. To do that we remove 50% of the edges for each network and then run the embedding algorithms. We form a testing set of the removed edges along with an equal number of randomly sampled non-edges. Then we compute $\boldsymbol{e}_i^T \boldsymbol{e}_j$ for each i, j edge in the testing set and rank the edges according to $\boldsymbol{e}_i^T \boldsymbol{e}_j$. Higher ranked edges are more likely to have a link. To assess the performance of the baselines we measure the area under ROC curve (AUC) and Average Precision (Avg. Prec.). The results are presented in Table 7 and are averaged over 5 shuffles. We observe that for `Wikipedia`, the proposed `GAGE` and the autoencoders work similarly and `AGE` is the best. In the `WebKB` network `DANE` works the best, whereas in `BlogCatalog` `Graph-VAE` and `Graph-AE` outperform `GAGE` and the baselines. However, taking into consideration that in node classification task `GAGE` works markedly better, we conclude that `GAGE` produces more informative node embeddings.

B SENSITIVITY ANALYSIS

In this subsection we examine the effect of parameter λ in the performance of the proposed `GAGE` embeddings for node classification and link prediction.

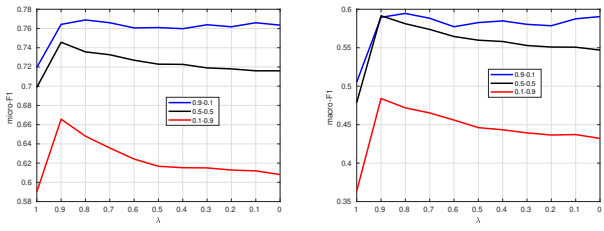
First, we test the effect of λ on node classification. We set the embedding dimension equal to $F = 128$ and vary λ from 1 to 0 with step equal to 0.1. We measure micro-F1 and macro-F1 scores for $90-10$, $50-50$ and $10-90$ training-testing splits. Recall that high values of λ aim to preserve the network geometry associated with the connectivity information, whereas low values of λ better preserve the attribute distances. The results for `Wikipedia`, `WebKB` and `BlogCatalog` are presented in Figs. 1, 3 and 5 respectively. The classification performance is consistent for $\lambda \in [0.1, 0.9]$ and the best performance is usually achieved for $\lambda \in [0.5, 0.9]$. When $\lambda = 1$ the focus is solely on the graph and classification performance is weaker compared to all other values. This stresses the importance of network attributes in node representation learning and graph node classification.

Next we examine the effect of parameter λ in link prediction. In this direction we vary λ from 1 to 0 with step equal to 0.1, as

Table 7: Average score and standard deviation over 5 shuffles for link prediction

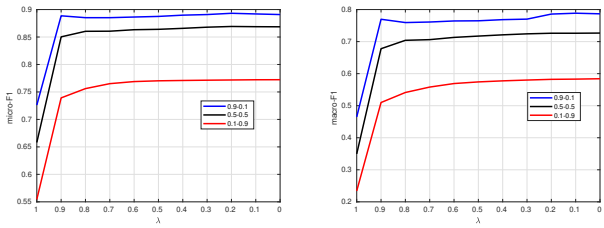
Algorithm	Dataset					
	Wikipedia		WebKB		BlogCatalog	
	AUC	Avg. Prec.	AUC	Avg. Prec.	AUC	Avg. Prec.
GAGE	0.8405 ± 0.0018	0.8819 ± 0.0017	0.8604 ± 0.0078	0.8347 ± 0.0112	0.7201 ± 0.0013	0.7589 ± 0.0077
T-PINE	0.8208 ± 0.0069	0.8767 ± 0.0044	0.7134 ± 0.0109	0.7308 ± 0.0121	0.6472 ± 0.0056	0.6638 ± 0.0034
Deepwalk	0.7965 ± 0.0056	0.8254 ± 0.0044	0.6104 ± 0.0145	0.6646 ± 0.0128	0.6645 ± 0.0049	0.6891 ± 0.0064
Graph-AE	0.8250 ± 0.0040	0.8833 ± 0.0043	0.8037 ± 0.0287	0.8335 ± 0.0225	**0.8231** ± 0.0222	0.8202 ± 0.0367
Graph-VAE	0.8479 ± 0.0073	0.8949 ± 0.0047	0.8014 ± 0.0375	0.8314 ± 0.0284	0.8218 ± 0.0100	**0.8248** ± 0.0164
TADW	0.7087 ± 0.0028	0.7722 ± 0.0032	0.7966 ± 0.0160	0.8178 ± 0.0186	0.5351 ± 0.0014	0.5317 ± 0.0008
DGI	0.8262 ± 0.0020	0.8409 ± 0.0019	0.7778 ± 0.0045	0.8179 ± 0.0032	0.7434 ± 0.0021	0.7404 ± 0.0003
AGE	**0.9173** ± 0.0025	**0.9151** ± 0.0048	0.9040 ± 0.0037	0.8612 ± 0.0098	0.7747 ± 0.0102	0.7533 ± 0.0097
DANE	0.8228 ± 0.0032	0.8346 ± 0.0025	**0.9201** ± 0.0100	**0.8715** ± 0.0088	0.6347 ± 0.0220	0.6526 ± 0.0378

Figure 1: Effect of λ on Wikipedia node classification



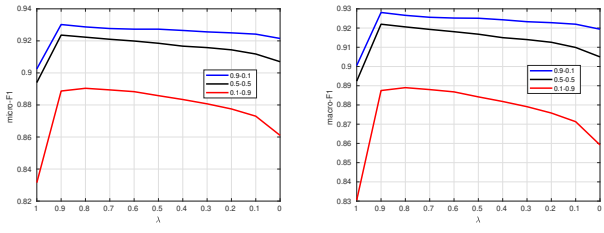
(a) micro-F1 score **(b) macro-F1 score**

Figure 3: Effect of λ on WebKB node classification



(a) micro-F1 score **(b) macro-F1 score**

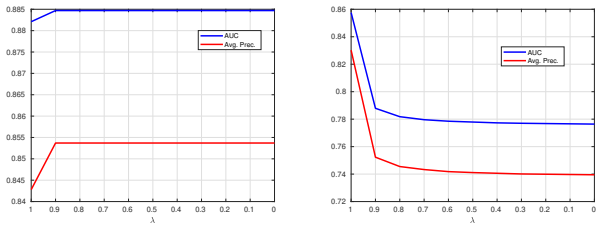
Figure 5: Effect of λ on BlogCatalog node classification



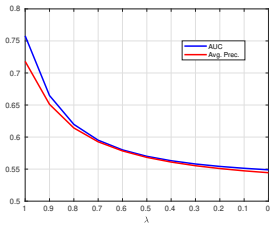
(a) micro-F1 score **(b) macro-F1 score**

before, and measure the AUC and Average Precision. The embedding dimension is set to $F = 64, 128, 256$ for WebKB, Wikipedia and BlogCatalog respectively. The results are presented in Fig. 7. For BlogCatalog the performance is consistent across all values of λ.

Figure 7: Effect of λ on link prediction



(a) Wikipedia **(b) WebKB**



(c) BlogCatalog

For Wikipedia and WebKB we observe that better link prediction is achieved when $\lambda = 1$ and the performance deteriorates as lambda decreases. This expected as potential links affect the graph geometry of the network and with $\lambda = 1$ we focus on preserving the connectivity distances between the nodes.