Node Proximity Is All You Need: Unified Structural and Positional Node and Graph Embedding

Jing Zhu*† Xingyu Lu*† Mark Heimann‡ Danai Koutra*

Abstract

While most network embedding techniques model the relative positions of nodes in a network, recently there has been significant interest in *structural embeddings* that model node *role equivalences*, irrespective of their distances to any specific nodes. We present PhUSION, a proximity-based unified framework for computing structural and positional node embeddings, which leverages well-established methods for calculating node proximity scores. Clarifying a point of contention in the literature, we show which step of PhUSION produces the different kinds of embeddings and what steps can be used by both. Moreover, by aggregating the PhUSION node embeddings, we obtain graph-level features that model information lost by previous graph feature learning and kernel methods. In a comprehensive empirical study with over 10 datasets, 4 tasks, and 35 methods, we systematically reveal successful design choices for node and graph-level machine learning with embeddings.

1 Introduction

Node embeddings model node similarities in a multi-dimensional feature space: the more similar two nodes are in a network, the closer they lie in this space. Two broad categories of node similarity are prevalent in the literature: (i) positional proximity, which embeds close nodes similarly [1]; and (ii) structural similarity, which embeds nodes similarly if they have similar roles or patterns of interaction with other nodes, irrespective of their relative locations [2]. In turn, these similarities lead to *positional* or *proximity-preserving* embeddings, and *structural* or *role-based* embeddings, respectively.

Characterizing the relationship between proximity-preserving and structural node embeddings is an open and contested problem, with recent works making opposing claims. For instance, Rossi et al. characterize these classes of methods as fundamentally different methodologically and in terms of applications [3]. Meanwhile, concurrent work proposed a *theoretical* framework in which the analogous concepts are actually equivalent for downstream tasks [4]. However, according to [3], it is unclear how this theoretical framework maps onto real-world graph mining methods.

A seminal work, NetMF [5], showed that various positional node embeddings amount to the same embedding technique (matrix factorization) applied to various matrices capturing pairwise node proximity scores. Going further, we propose PhUSION, a proximity-based unified framework for computing structural *and* positional node embeddings. PhUSION has three steps: (i) computation of pairwise node proximities, (ii) application of a nonlinear filter, and (iii) application of a dimensionality-reducing embedding function. We show which steps can be used for proximity-preserving or structural embedding and which step makes them different, revealing similarities and differences between the two classes of methods.

Additionally, PhUSION generalizes existing methods and yields novel ones from 35 different combinations of design choices, some of which improve on the variations studied in the literature. We extensively perform an empirical study of possible design choices for both structural and proximity-preserving node embeddings, to understand what works and why. In particular, nonlinear filtering has very recently been identified [6] as a key ingredient to the success of proximity-preserving node embedding. We analyze this observation in much greater detail for proximity-preserving embeddings and for the first time apply it to structural embeddings.

We extend PhUSION to embed entire graphs, a problem for which separate solutions have been proposed using graph signatures and similarity scores derived from node proximity matrices [7, 8] and aggregated structural node embeddings [9]. Since we have shown that node proximity matrices can be used to derive structural node embeddings, we interpret previous methods [7, 8] as embedding aggregation; we use PhUSION to learn more expressive graph features by ag-

*Computer Science & Engineering, University of Michigan. Email: {jingzhuu, luxingyu, mheimann, dkoutra}@umich.edu

†Authors contributed equally to this work.

‡Lawrence Livermore National Laboratory. Work partially completed while a student at the University of Michigan.

gregating our more informative node embeddings, that model information we show that previous works cannot.

Our contributions are summarized as follows:

- **Unifying Embedding Perspective**: We propose PhUSION, which can use pairwise node proximity matrix to generate embeddings that model node similarity based on structural roles or positional proximity. Our analysis of PhUSION shows the technical similarities and differences between structural and proximity-preserving node embeddings, a contested open question [3, 4].
- **Study of Successful Design Choices**: On benchmark tasks for proximity-preserving and structural embedding choices, we investigate the combination of node proximity matrices, nonlinear transformation, and embedding functions. Our results uncover new insights that can improve both proximity-preserving and structural embeddings.
- **Graph-Level Learning**: We turn PhUSION into a method for learning features for entire networks from their node proximity matrices based on node embedding aggregation. We interpret previous graph kernels [8] and feature learning methods [7] as simplified versions of PhUSION, and show what information we can capture with more expressive design choices that these previous works cannot.

We provide code and additional supplementary material at `https://github.com/GemsLab/PhUSION`.

2 Related Work

Frameworks for Node Embedding. Node embeddings are latent feature vectors for nodes in a network that are similar for similar nodes. Most embedding methods define node similarity in terms of **proximity** (e.g. direct or indirect connection) within a single graph. In contrast, **structural** embedding methods capture a node's structural role independent of its proximity to specific nodes; this independence makes embeddings comparable across distant parts of a graph [10] or separate graphs [11, 9]. Both kinds of embeddings may be obtained using a diverse range of shallow and deep learning methods. For more information, we refer the reader to a survey [1] on proximity-preserving or positional embeddings, and a recent comprehensive empirical study on structural or role-based embeddings [10].

The plethora of node embedding methods has raised interest in finding unifying frameworks for different methods, which can also lead to new technical advances. For example, many proximity-preserving embedding methods were shown to implicitly factorize different proximity-based node similarity matrices; this insight inspired the NetMF method based on explicit matrix factorization [5]. It is known that many (proximity-preserving) node embedding methods can be summarized as a two-step process of node similarity matrix construction and dimensionality reduction [12]. However, PhUSION is the first framework to subsume both proximity-preserving and structural embedding methods. Moreover, in light of recent work [6], we carefully study a third step of applying a nonlinearity before performing dimensionality reduction.

Graph Comparison. For comparing entire graphs, aggregating node embeddings (as we do) is competitive to deep neural networks, graph kernels, and feature construction [9]. Because a graph's node proximity matrix captures important information, many works have sought to use this *within*-graph information for *cross*-graph comparison. A challenge is that nodes in different graphs may not correspond. Feature learning method NetLSD [7] and graph kernel RetGK [8] solve this problem by only considering node self-similarities, which forgoes directly modeling a node's similarity to other nodes (cross-*node* similarities). Other graph similarity functions such as DeltaCon [13] model cross-node similarities, but are restricted to graphs defined on the same set of nodes. However, PhUSION can model within-graph cross-node similarities for more expressive general cross-graph comparison.

3 Unified Theoretical Framework

In this section, we present the abstract steps of our PhUSION framework for node and graph feature learning, before describing concrete choices in the next section.

Preliminaries. We consider a graph G with node set V and adjacency matrix $\mathbf{A}$ containing edges between nodes. We learn an $n \times d$ matrix $\mathbf{Y}$ of d-dimensional node embeddings, where the i-th row $\mathbf{Y}_i$ is a feature representation for node i. For ease of reference, we define common quantities for graph learning and node embedding, along with parameters specific to certain node proximity functions, in Tab. 1.

Structural vs Positional Embeddings. Structural node embedding should learn similar features for automorphically equivalent or near-equivalent nodes [10, 4], even if they are distant from each other in the network. On the other hand, for nodes to have similar positional embeddings, they must be close in the network. Although these are two very different embedding outcomes, the steps we present below can generate either kind of embedding; later, we will show concretely where the difference arises.

3.1 Node Feature Learning

For learning node features from a graph with adjacency matrix $\mathbf{A}$, we perform the following three steps:

Table 1: Symbols and definitions

Symbol		Definition
Standard graph matrices	$\mathbf{A}$	Adjacency matrix
	$\mathbf{D}$	Diagonal matrix of node degrees
	$\mathbf{L}$	Unnormalized Laplacian matrix ($\mathbf{D}-\mathbf{A}$)
	$\mathbf{L}^+$	Pseudoinverse of $\mathbf{L}$
	$\mathbf{R}$	Random walk transition matrix ($\mathbf{D}^{-1}\mathbf{A}$)
	k	Matrix power
PhUSION functions	$\Psi()$	Node proximity function
	$\sigma()$	Nonlinear transformation function
	$\zeta()$	Embedding function
	$\mathbf{S}$	Matrix of node proximities $\mathbf{S}=\Psi(\mathbf{A})$
	$\tilde{\mathbf{S}}$	Matrix of nonlinearly filtered node proximities $\tilde{\mathbf{S}}=\sigma(\Psi(\mathbf{A}))$
	$\mathbf{Y}$	Matrix of node embeddings $\mathbf{Y}=\zeta(\sigma(\Psi(\mathbf{A})))$
PPMI [5]	vol(G)	$\Sigma_{i,j}\mathbf{A}_{ij}$
	T	Window size
	b	Parameter for negative sampling
Heat kernel [2, 7]	g_s	Filter kernel with scaling parameter s
	$\mathbf{\Lambda}$	Diagonal matrix of eigenvalues of $\mathbf{L}$
	$\mathbf{U}$	Eigenvectors of $\mathbf{L}$ ($\mathbf{L}=\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$)
FaBP [14]	h_h	$\sqrt{(-c_1+\sqrt{c_1^2+4c_2})/8c_2}$, where c_1 =trace($\mathbf{D}$) + 2; c_2 =trace($\mathbf{D}$) − 1
	a	$4h_h^2/(1-4h_h^2)$
	c	$2h_h/(1-4h_h^2)$
PPR [15]	β	Decay parameter

Step 1: Calculate node proximities $\mathbf{S}$ using a function $\Psi(\mathbf{A})$;

Step 2: Filter these proximities via a nonlinearity function $\tilde{\mathbf{S}}=\sigma(\mathbf{S})$; and

Step 3: Embed the transformed proximities using a dimensionality reduction function: $\mathbf{Y}=\zeta(\tilde{\mathbf{S}})$.

Our node embedding framework can be precisely summarized by function composition:

$$\mathbf{Y}=\zeta(\sigma(\Psi(\mathbf{A}))) \tag{3.1}$$

Multiscale Node Embeddings. Many proximity functions can be tuned with scaling parameters to capture more local or global proximity [2, 16]. We can create multiscale embeddings by concatenating embeddings using the same node proximity function at several different scales:

$$\mathbf{Y}=||_i\mathbf{Y}^{(s_i)}=\mathbf{Y}^{(s_1)}||\mathbf{Y}^{(s_2)}||\dots||\mathbf{Y}^{(s_t)}, \tag{3.2}$$

where embeddings at each individual scale are computed with Eq. (3.1) using the desired scale parameter to compute node proximity: $\mathbf{Y}^{(s_i)}=\zeta(\sigma(\Psi(\mathbf{A};s_i)))$.

3.2 Graph Feature Learning We can aggregate a graph's node embeddings into a single feature vector that describes the entire graph using a function $\rho()$:

$$\mathbf{f}=\rho(\mathbf{Y}) \tag{3.3}$$

4 Unifying Node Embedding Methods

We now propose concrete function choices for Eqs. (3.1)-(3.3), and characterize general and specific choices.

4.1 Step 1: Computing Node Proximities $\Psi()$. The first step of our framework, PhUSION, is to create a matrix of pairwise node proximities $\mathbf{S}\in\mathbf{R}^{n\times n}$. $\mathbf{S}_{ij}$ should be large for nodes that are close in the graph (e.g. neighbors) and small for faraway nodes. Different proximity matrices have been used not only for node embedding but throughout graph mining, including:

- Positive pointwise mutual information (`PPMI`) [5]: $\mathbf{S}=\frac{\text{vol}(G)}{bT}(\sum_{r=1}^{T}\mathbf{R}^r)\mathbf{D}^{-1}$.
- Heat kernel (`HK`) [2]: $\mathbf{S}=\mathbf{U}g_s(\mathbf{\Lambda})\mathbf{U}^\top$.
- Belief Propagation (`FaBP`) [14]: $\mathbf{S}=(\mathbf{I}+a\mathbf{D}-c\mathbf{A})^{-1}$.
- Personalized Pagerank (`PPR`) [15]: $\mathbf{S}=(\mathbf{I}-\beta\mathbf{A})^{-1}(\beta\mathbf{A})$.
- Laplacian pseudoinverse ($\mathbf{L}^+$) [6]: $\mathbf{S}=\mathbf{L}^+$, which approximates the PPMI matrix as the window size $T\to\infty$, up to a low-rank correction term.
- Powers of the adjacency matrix (`Adj`) [15, 16] or random walk matrix (`RW`) [8]: $\mathbf{S}=\mathbf{A}^k$ or $\mathbf{S}=\mathbf{R}^k$.

4.2 Step 2: Nonlinear Transformations of Node Proximities $\sigma()$. As a preprocessing step before embedding, we can filter the node proximities with a nonlinear function $\sigma(\mathbf{S})$. Recent work [6] argues that such nonlinearity is largely responsible for the performance gain of recent deep learning-inspired node embedding methods. Thus, we consider the following functions:

- No nonlinearity: $\sigma(\mathbf{S})=\mathbf{S}$ (`Identity` function).
- Elementwise logarithm (`Log`): For proximity-preserving embedding with `PPMI`, we set $\sigma(\mathbf{S})_{i,j}=\log(\max\{\mathbf{S}_{i,j},1\})$ [5]. For other matrices with values concentrated in $[0,1]$, we propose to keep more information by only filtering out negative or zero elements:

$$\sigma(\mathbf{S})_{i,j}=\begin{cases}0 & ,\mathbf{S}_{i,j}\le 0\\ \log(\dfrac{\mathbf{S}_{i,j}}{\min(\mathbf{S}^+)}) & ,\mathbf{S}_{i,j}>0\end{cases}$$

  where $\min(\mathbf{S}^+)$ is the smallest positive element in $\mathbf{S}$.
- Thresholded binarization (`Bin-p`) [6]: Let $a\in\mathbf{N}$ be the p-th percentile (p% smallest element) in $\mathbf{S}$. Then $\sigma(\mathbf{S})$ is defined elementwise as:

$$\sigma(\mathbf{S})_{i,j}=\begin{cases}0 & ,\mathbf{S}_{i,j}\le a\\ 1 & ,\mathbf{S}_{i,j}>a\end{cases}$$

4.3 Step 3: Embedding Node Proximities $\zeta()$. Given a (filtered) similarity matrix $\tilde{\mathbf{S}}$, node embeddings learn low-dimensional feature representations using various dimensionality reduction techniques. We represent the embedding process as a function $\zeta(\tilde{\mathbf{S}})$.

- One way to generate d-dimensional embeddings is by factorizing the node similarity matrix, prototypically

with singular value decomposition (SVD) [5]. Based on rank-d SVD $\tilde{\mathbf{S}} \approx \mathbf{U}_d\mathbf{\Sigma}_d\mathbf{V}_d$, we can obtain the node embeddings as $\zeta(\tilde{\mathbf{S}}) = \mathbf{U}_d\mathbf{\Sigma}_d^{\frac{1}{2}}$.

- Another way to generate a d-dimensional embeddings from an $n \times n$ similarity matrix $\tilde{\mathbf{S}}$ is characteristic function sampling (CFS). For even dimensionality d, we compute the embedding of each node u by sampling real and imaginary components from its empirical characteristic function, $\phi_u(t) = \sum_{v=1}^{n} \exp(it\tilde{\mathbf{S}}_{vu})$, evaluated at $\frac{d}{2}$ evenly spaced landmarks $t_1, \ldots, t_{d/2}$ between 0 and 100 [2]. CFS is a permutation-invariant function applied row-wise to $\tilde{\mathbf{S}}$ that models the distribution of a node's proximity scores [2].

Special Cases. PhUSION generalizes several existing proximity-preserving and structural embedding methods, which we summarize in the following result:

THEOREM 4.1. *Special cases of Eq.* (3.2) *include but are not limited to: GraphWave [2], NetMF [5], InfiniteWalk [6], HOPE [15], GraRep [16], DNGR [17], and sRDE [18] for signed networks.*

Proof. We give the constructions in App. A.1. ∎

4.4 What Makes Node Embeddings Positional or Structural? We isolate the embedding function $\zeta()$ as the responsible design choice for making PhUSION yield positional or structural embeddings. Concretely, embedding a proximity matrix using SVD produces positional embeddings, while using CFS (or any other permutation-invariant row function) produces structural embeddings. On the other hand, any choice of $\Psi()$ and $\sigma()$ can yield positional or structural embeddings.

THEOREM 4.2. *Let connected graphs G_1, G_2 have an isomorphism $\pi : V_1 \to V_2$, i.e. a bijective mapping between the nodes and $\mathbf{A}_2 = \mathbf{P}\mathbf{A}_1\mathbf{P}^\top$, where the binary matrix $\mathbf{P}$ has nonzero elements exactly at the entries $(\pi(i), i)$ for $i \in [1, \ldots, |V|]$. Define a combined graph G with block diagonal adjacency matrix $\mathbf{A} = [\mathbf{A}_1, \mathbf{0}; \mathbf{0}, \mathbf{A}_2]$, so that π encodes an automorphism within G. Assume that node proximity and nonlinearity functions $\Psi()$ and $\sigma()$ preserve this automorphism: $\tilde{\mathbf{S}}_2 = \mathbf{P}\tilde{\mathbf{S}}_1\mathbf{P}^\top$, where $\tilde{\mathbf{S}}_i = \sigma(\Psi(\mathbf{A}_i))$. Also assume that disconnected nodes have proximity score 0 (unchanged by nonlinearity), so that $\tilde{\mathbf{S}} = \sigma(\Psi(\mathbf{A})) = [\tilde{\mathbf{S}}_1, \mathbf{0}; \mathbf{0}, \tilde{\mathbf{S}}_2]$. Let $\mathbf{Y}$ be the combined embeddings of G, which can be split into embeddings $\mathbf{Y}^{(1)}$ and $\mathbf{Y}^{(2)}$ corresponding respectively to the nodes originally in G_1 and G_2. Then:*

1. *If $\mathbf{Y} = SVD(\tilde{\mathbf{S}})$, then $\mathbf{Y}^{(1)}_i \neq \mathbf{Y}^{(2)}_{\pi(i)}$.*
2. *If $\mathbf{Y} = CFS(\tilde{\mathbf{S}})$, or more generally any permutation-invariant function $\zeta(\tilde{\mathbf{S}})$, then $\mathbf{Y}^{(1)}_i = \mathbf{Y}^{(2)}_{\pi(i)}$.*

Proof. See supplementary App. A.2. ∎

Note: Some existing methods learn structural embeddings with implicit or explicit matrix factorization [19, 11], which in PhUSION would produce positional embeddings. The key difference is that these methods do not factorize a pairwise node proximity matrix, but a *structural* similarity matrix (where disconnected nodes may have a nonzero similarity score). One advantage of PhUSION is that the node proximity matrices we use are well studied throughout graph mining.

5 Unifying Graph Embedding Methods

Our PhUSION framework also produces features that describe an entire graph, when we aggregate its nodes' embeddings into a single feature vector. Here, we show that two recent graph kernels and feature maps are in essence special cases of PhUSION.

PhUSION:NetLSD. NetLSD computes graph features from its heat kernel matrix at multiple scales [7]. For scales $s_1, \ldots, s_d$, the resulting d-dimensional feature vector has as its i-th entry $h^{(s_i)}$, the trace of the heat kernel matrix at scale s_i. For size invariance, the authors propose normalizing an n-node graph's features by the heat trace of the n-node empty graph, which amounts to multiplying by $\frac{1}{n}$. Thus, the exact normalized NetLSD features are: $\frac{1}{n}[h^{(s_1)}, \ldots, h^{(s_d)}]$.

THEOREM 5.1. *NetLSD (using the heat kernel with empty graph normalization) is a special case of Eq.* (3.3) *where $\Psi()$ computes the graph's heat kernel matrix at multiple scales s as its proximity matrix $\mathbf{S}$, $\zeta(\mathbf{S}) = diag(\mathbf{S})$, $\sigma()$ is the identity function, and $\rho()$ averages the embeddings.*

Proof. At scale s_k, the one-dimensional node embedding of node i is given by $\mathbf{y}_i^{(s_k)} = \mathbf{S}_{ii}^{(s_k)}$. Thus, for d scales $s_1, \ldots, s_d$, the multiscale embedding of node i given by Eq. (3.2) is $\mathbf{y}_i = [\mathbf{S}_{ii}^{(s_1)}, \ldots, \mathbf{S}_{ii}^{(s_d)}]$. Aggregating these node features into graph features using Eq. (3.3) gives $\mathbf{f} = \frac{1}{n}\sum_i \mathbf{y}_i = \frac{1}{n}[\sum_i \mathbf{S}_{ii}^{(s_1)}, \ldots, \sum_i \mathbf{S}_{ii}^{(s_d)}] = \frac{1}{n}[\mathrm{Tr}(\mathbf{S}^{(s_1)}), \ldots, \mathrm{Tr}(\mathbf{S}^{(s_d)})]$. When $\mathbf{S}$ is the heat kernel matrix, each term becomes $\mathrm{Tr}(\mathbf{S}^{(s_i)}) = h^{(s_i)}$. ∎

PhUSION:RetGK. The scalable graph kernel ($\mathrm{RetGK_{II}}$) [8] based on approximate feature maps [20] is defined as $K(G_1, G_2) = \kappa\Big(\bar{\mathbf{f}}(G_1), \bar{\mathbf{f}}(G_2)\Big)$. Without node attributes, $\bar{\mathbf{f}}(G) = \sum_{i=1}^{n} \phi(\mathbf{y}_i)$ where the j-th entry of $\mathbf{y}_i$ is the return probability of a random walk of length j starting from node i (formally $\mathbf{R}_{ii}^j$), and ϕ is a feature map approximating a vector-valued kernel [20]. It can thus be seen that RetGK has essentially the same form as the other methods:

THEOREM 5.2. *Without node attributes and with ϕ and κ both set to the linear kernel, RetGK is a special case of Eq.* (3.3) *where: for multiple values of the parameter s, $\Psi()$ computes the graph's s-step random walk transition matrix as its proximity matrix $\mathbf{S}$, $\zeta(\mathbf{S}) = diag(\mathbf{S})$, $\sigma()$ is the identity function, and $\rho()$ averages the embeddings.*

In practice, [8] proposes to set ϕ to be a random Fourier feature map to approximate the Gaussian kernel [20], and κ to be a Gaussian or Laplace kernel, applying the successive embedding trick used for graph kernels [21]. Node attributes may be incorporated by taking the Kronecker product of the attribute vectors with the embeddings [8]. All of these techniques readily apply to any of the other methods we have proposed.

Expressive Graph Comparison with PhUSION. Postprocessing aside, we can interpret RetGK and NetLSD as instances of PhUSION: they average multiscale embeddings learned from different node proximity matrices (`HK` for NetLSD, `RW` for RetGK). However, they use a 1-dimensional embedding function mapping nodes to their corresponding diagonal elements in $\mathbf{S}$. Of course, this simple embedding loses off-diagonal information in $\mathbf{S}$ (namely, inter-node proximities), which our embeddings capture. To show the greater expressivity of our embeddings $\mathbf{Y}$ by a fair comparison, we also use mean pooling for our $\rho(\mathbf{Y})$, although more complex aggregation functions could be used [9].

6 Experiments

To extensively evaluate PhUSION in a variety of contexts, we consider several real datasets for node classification (Tab. 2a) for which positional and structural role-based embeddings have been shown to be most effective (§ 6.1). For the latter, we also use synthetic data exhibiting clear role equivalences, the structure of which we can precisely control [2, 10]. We also evaluate aggregated structural embeddings for graph classification (§ 6.2) on real benchmark datasets (Tab. 2b).

6.1 Node-level Embedding. First we evaluate PhUSION in the node classification task with positional and structural node embeddings.

Setup. We combine 7 node proximity functions $\Psi()$ and 5 different nonlinearities $\sigma()$ (including `Identity`). Following our theoretical analysis (§4.4), we use SVD to generate positional node embeddings and CFS to generate structural embeddings. In total, the PhUSION framework gives us **35 different node embedding methods** of each type, including positional embeddings NetMF [5], InfiniteWalk [6], and HOPE [15] and structural embedding method GraphWave [2] as special cases. We tune hyperparameters with grid search and report the procedure and best parameters in App. B. Interestingly, we find that the best parameters strongly model local node proximity.

We follow the supervised machine learning setup of [19]: we randomly sample 80% of the dataset for training and the rest for testing. For multi-label prediction, we use the one-vs-rest logistic regression model [5] and evaluate using micro-F1 scores.

Table 2: Real Datasets

(a) Node Classification

	Dataset	# Nodes	# Edges	Labels
Proxim.	**BlogCatalog** [5]	10,312	333,983	Blogger Interests (39)
	PPI [5]	3,890	76,584	Biological states (50)
	Wikipedia [5]	4,777	184,812	Part-of-Speech tags (40)
Struct.	**Brazil** [19]	131	1,038	# landings & take-off (4)
	Europe [19]	399	5,995	# landings & take-off (4)
	USA [19]	1,190	13,599	# passengers (4)

(b) Graph Classification

Dataset	# Graphs	Avg # Nodes	Labels
IMDB-M [22]	1,500	13.00	Collaboration genre (3)
PROTEINS [22]	1,113	39.06	Protein type (2)
PTC-MR [22]	344	14.29	Molecular property (2)

6.1.1 Positional Node Embedding. We report raw results for all 35 positional node embedding methods derived from PhUSION in Fig. 1. Table 3 performs a drilldown on a per-design choice basis.

Results. We can see that `PPMI` does an excellent job, while $\mathbf{L}^+$ is also competitive. As for the nonlinearity $\sigma()$, our findings support recent work [6] that adding nonlinearity is a critical part of outperforming the original spectral embedding approaches: it is almost always beneficial for all proximity matrices. On average, we find that `Log` does the best; however, `Bin-p` also performs better than `Identity` (no nonlinearity), and indeed the best embedding method for two of the three datasets (PPI and Wikipedia) uses binarization.

The use of binarization as nonlinearity and $\mathbf{L}^+$ for proximity was proposed by InfiniteWalk [6], and the use of `PPMI` node proximities with `Log` nonlinearity is the NetMF method [5]. Our findings confirm that these recently identified design choices are indeed among the most successful overall. However, new design choices are competitive with them and may warrant further exploration. Moreover, no single choice of nonlinearity function $\sigma()$ performs best, nor does performance vary monotonically with the sparsity of the resulting matrix (`Bin-50` performs better than both `Bin-5` and `Bin-95`). Corroborating [6], deeper characterization of various choices of $\sigma()$ and their effects is of continued interest.

OBSERVATION 1. *(1) Nonlinearity has a complex effect,*

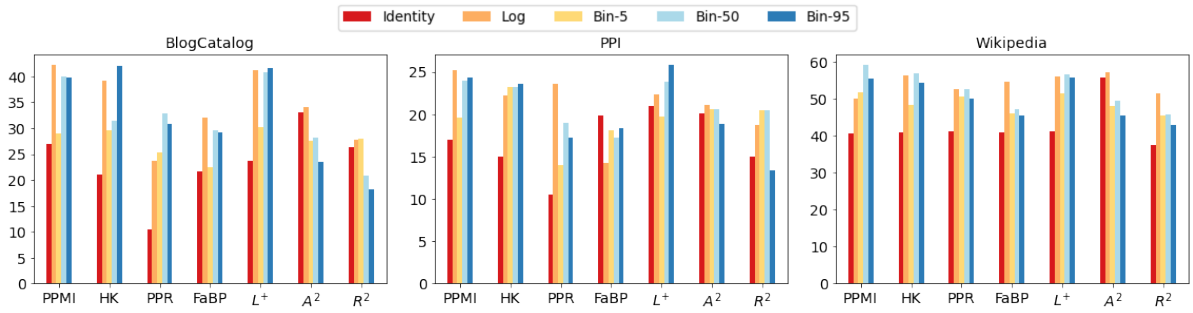


Figure 1: Node classification performance (micro-F1 scores) with positional embedding. Nonlinearity generally helps, but the best nonlinearity function varies across proximity matrices, and the best proximity matrix varies across datasets.

Table 3: Average rank and average/max micro-F1 scores of different proximity $\Psi()$ and nonlinearity functions $\sigma()$ on all datasets used for positional node embedding. Design choices used in existing methods NetMF and InfiniteWalk perform well on average (better than HOPE , which uses various $\Psi()$ functions but no nonlinearity). However, new design combinations are competitive.

		BlogCatalog			PPI			Wikipedia		
		Avg Rank	Avg Acc	Max Acc	Avg Rank	Avg Acc	Max Acc	Avg Rank	Avg Acc	Max Acc
$\Psi()$	PPMI	10.4	35.56	**42.21**	10.8	22.03	25.25	14.2	51.41	**59.10**
	HK	13.2	32.66	41.99	12.0	21.44	23.61	13.4	51.33	56.89
	PPR	21.6	24.61	32.80	23.8	16.84	23.63	17.2	49.38	52.69
	FaBP	20.6	26.94	31.98	24.8	17.57	19.89	22.2	46.79	54.43
	$\mathbf{L^+}$	10.0	35.49	41.55	8.8	22.52	**25.80**	11.8	52.11	56.47
	$\mathbf{A^2}$	17.2	29.25	34.06	15.2	20.21	21.04	14.6	51.13	57.01
	$\mathbf{R^2}$	26.0	24.25	28.04	23.0	17.61	20.48	25.4	44.57	51.52
$\sigma()$	Identity	25.43	23.31	33.04	24.0	16.9	20.93	27.86	42.53	55.82
	Log	10.86	34.30	**42.21**	12.86	21.07	25.25	8.86	53.97	57.01
	Bin-5	20.43	27.44	30.14	18.71	19.39	23.23	19.14	48.76	51.77
	Bin-50	14.0	31.95	40.85	12.71	21.16	23.97	11.57	52.53	**59.10**
	Bin-95	14.29	32.11	41.99	16.29	20.21	**25.80**	17.43	49.86	55.63

but is essential in improving the performance of positional node embedding.

(2) Generally, design choices identified by recent works [5, 6] are among the most successful across datasets, but new combinations are often competitive.

6.1.2 Structural Node Embedding. We now evaluate the 35 methods we obtain from the PhUSION framework for structural role-based node embedding in two major tasks, node classification and clustering.

Node Classification. We again perform supervised machine learning to predict the node labels from the node embeddings, but in this case on datasets where the labels correspond to nodes' structural roles. We plot the accuracy of each combination of design choice in Fig. 2, and the average rank, mean and maximum accuracy of each individual design choice in Tab. 4.

Node Clustering. Following the literature on structural node embedding [2, 10], we also assess our methods using networks that are constructed to manifest distinctive structural roles. Our goal is to cluster nodes with similar structural roles. We follow the dataset construction (cf. App. C) and clustering setup of [2]. These datasets exhibit clear role equivalence (perturbed by noise). For brevity, we only report results from embeddings without nonlinearity. We assess the clustering quality using homogeneity, completeness, and silhouette score.

Results. *Node Classification.* We see different trends than positional node embeddings. In this case, nonlinearity is not always helpful; indeed `Identity` is on average much more competitive. However, all datasets, using another proximity method or nonlinearity improves on GraphWave as proposed, highlighting the flexibility of PhUSION. We find that a very simple nonlinearity, binarization, produces the best methods on two datasets: as CFS models the distribution of entries in each row, embedding a binary distribution simply models how many large proximities a node has to other

Table 4: ***Real data (left)*****: Average rank and average/max accuracy of different proximity $\Psi()$ and nonlinearity $\sigma()$ functions on all datasets used for structural node embedding.** ***Synthetic data (right)*****: Averaged clustering results for synthetic data with planted structural roles. For both tasks, we can dramatically improve on GraphWave by using a different proximity matrix and/or nonlinearity.**

		Brazil			Europe			USA			Synthetic		
		Avg Rank	Avg Acc	Max Acc	Avg Rank	Avg Acc	Max Acc	Avg Rank	Avg Acc	Max Acc	Hom	Comp	Silh
$\Psi()$	PPMI	28.00	37.72	43.48	29.80	37.06	46.82	25.80	43.71	54.13	.5283	.5029	.4986
	HK	6.80	68.75	**72.37**	7.20	52.65	54.45	7.20	58.96	**63.49**	.5951	.5488	.4392
	PPR	20.20	53.04	63.41	22.00	45.43	50.07	25.60	43.55	51.16	.5951	.5973	**.9307**
	FaBP	19.80	52.70	70.15	23.00	44.94	49.90	22.00	47.65	56.92	**.7157**	.6627	.5531
	$\mathbf{L}^+$	27.60	39.18	53.41	15.00	46.42	**56.02**	24.00	44.02	59.94	.2071	.1896	.2499
	$\mathbf{A}^2$	9.80	64.03	71.85	12.80	50.27	53.97	9.80	57.67	59.83	**.7156**	**.6750**	.5760
	$\mathbf{R}^2$	13.40	63.01	67.56	16.00	48.97	51.80	11.40	56.56	58.56	.6551	.6071	.4232
$\sigma()$	Identity	14.23	60.33	71.78	12.57	50.71	**56.02**	13.43	54.28	59.95			
	Log	18.43	54.60	71.85	21.71	44.19	53.65	16.14	52.01	62.73			
	Bin-5	20.14	50.68	71.85	20.14	44.90	51.58	19.43	48.61	60.71		N/A	
	Bin-50	11.85	60.78	**72.37**	13.14	49.21	54.68	17.14	51.94	**63.49**			
	Bin-95	25.57	43.91	62.96	22.29	43.67	54.45	23.86	44.68	56.97			

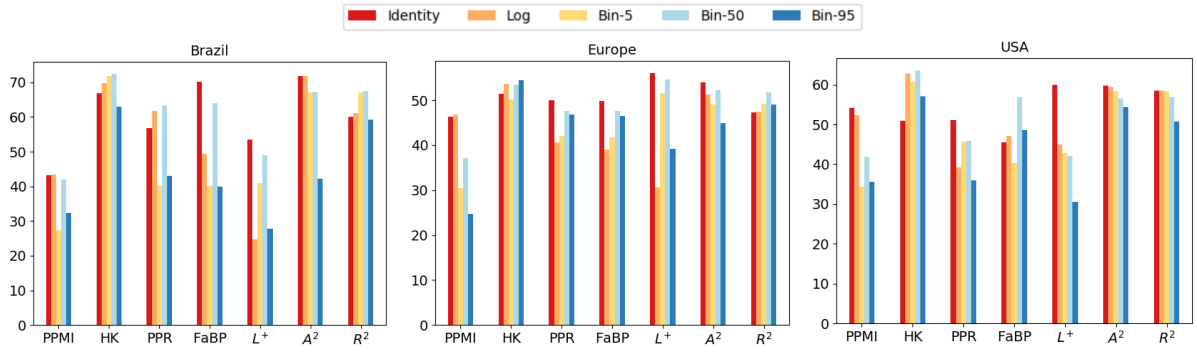


Figure 2: Node classification performance with structural embeddings. Many different proximity matrices and nonlinearity functions can yield high accuracy, often higher than existing method GraphWave.

nodes. This corroborates a recent claim [10] that simple structural information suffices for these datasets.

Node Clustering. The results in Tab. 4 (right) show that a variety of proximity matrices successfully cluster nodes by their structural roles, in some cases better than the heat kernel used in GraphWave [2]. We show similar results on unperturbed graphs in the supplementary § C.

Observation 2. *Within our PhUSION framework, we discover design choices for structural embedding that improve on downstream tasks compared to existing methods. In particular, we discover that some design choices used for positional node embeddings, like nonlinearity, can improve structural embeddings as well.*

6.1.3 Comparing Design Choices for Positional & Structural Embeddings. Based on all our node-level experiments, we see that although the same design choices prior to embedding $(\Psi(), \sigma())$ can be used for positional or structural embeddings, in practice the best design choices for each kind of embedding tend to be different. For instance, nonlinearity is almost always helpful for positional node embeddings, but only sometimes helpful for structural embeddings. Proximity functions PPMI and $\mathbf{L}^+$ tend to be successful for positional node embeddings, but do not produce the best structural embeddings (clearly seen on the clustering tasks).

This analysis raises an important question: Can we characterize node proximity matrices that produce good embeddings of either type? We perform initial exploratory analysis in App. E, investigating properties of the matrices produced by each combination of $\Psi()$ and $\sigma()$. We find that the row-wise sums of elements in matrices producing good positional node embeddings tend to have a bell-shaped distribution, whereas we observe power-law distributions in matrices that produce good structural embeddings.

Observation 3. *While positional and structural node*

Table 5: Graph classification using averaged node embeddings (Eq. 3.3) and baselines (gray). We improve on NetLSD (3/3 datasets) and RetGK (2/3 datasets), which leverage simpler features from HK and RW matrices, using our embeddings of these matrices. We may also use different proximity matrices like Adj, which can further increase performance.

	PPMI	FaBP	HK	PPR	Adj	RW	$\mathbf{L}^+$	NetLSD	RetGK
IMDB-M	38.98 ± 0.56	46.54 ± 0.27	48.18 ± 0.19	41.62 ± 0.07	$\mathbf{49.44 \pm 0.36}$	47.42 ± 0.32	45.44 ± 0.47	44.17 ± 0.05	43.91 ± 0.74
PROTEINS	70.50 ± 0.36	73.38 ± 0.19	73.94 ± 0.16	71.64 ± 0.08	72.36 ± 0.34	71.52 ± 0.17	70.76 ± 0.30	71.96 ± 0.04	$\mathbf{74.37} \pm 0.06$
PTC-MR	56.80 ± 0.40	55.48 ± 0.80	$\mathbf{59.18 \pm 0.97}$	58.84 ± 0.71	55.02 ± 0.77	58.22 ± 0.60	58.44 ± 0.59	58.84 ± 1.37	57.56 ± 1.27

embeddings may begin with the same node proximity designs, in practice the best designs for each kind of embedding method tend to differ.

This may be one reason why the survey work [3], characterizing existing examples of positional and structural node embedding methods, judged their methodology to be fundamentally different (even though our framework and the theory of [4] show a methodological connection in principle).

6.2 Graph-Level Embedding. We now investigate PhUSION's effectiveness in learning graph features from various node proximity matrices. Intuitively, we expect that our more expressive features will allow us to classify graphs more accurately than previous works.

Setup. Our experiments evaluate the graph classification accuracy on PTC-MR, IMDB-M and PROTEINS datasets [22]. As our focus is learning from the graph structure alone, we ignore node attributes. We only use CFS (i.e. structural embeddings), which are comparable across graphs [9], and do not use nonlinearity $\sigma()$ as the baselines do not. We use a linear SVM to predict graphs' labels from their features; we report the 10-fold cross-validation accuracy averaged over 5 trials [9].

We compare against NetLSD [7] and RetGK [8], alternative ways of deriving graph features from HK and RW proximity matrices, respectively (§5). We use NetLSD's default 250 heat kernel values logarithmically spaced in the range $\{10^{-2}, 10^2\}$. We run RetGK using its defaults of 50th-order random walk return probabilities and its proposed exact and approximate successive kernel embedding (κ and ϕ in §5). We describe our hyperparameter settings in supplementary App. B; we parallel the settings of NetLSD and RetGK, and carefully avoid giving ourselves any unfair advantage over them (in fact, they have a slight advantage if anything: we leave NetLSD with its default higher dimensionality and RetGK with its default successive kernel embeddings).

Results. In Tab. 5, we see that our methods generally improve on NetLSD and RetGK as a way of getting graph features from their node proximity matrices. In particular, embedding RW using Eq. 3.2 outperforms RetGK, which is also based on the RW proximity matrix, on two out of three datasets (PTC-MR and IMDB-M). Similarly embedding HK outperforms NetLSD, which also uses the heat kernel matrix, on all three datasets (and outperforms all other methods on two datasets). This is strong evidence that by modeling each node's full distribution of proximities rather than its self-proximity, PhUSION captures more useful information.

Because we keep the embedding dimension the same as (or lower) than NetLSD and RetGK, which capture only a single value for a node at each proximity scale (whereas we return a 10-dimensional embedding), we necessarily consider much fewer scales. Our good comparative performance indicates that modeling more graph information at fewer scales is generally superior to modeling less information at more scales.

OBSERVATION 4. *PhUSION gives us a way to learn graph features from a given node proximity matrix that yield greater accuracy than previous works [7, 8], likely because of their expressivity (§ 5).*

6.3 Additional Analysis. For all our classification tasks, we also study the effect of proximity order for multiscale embeddings in the supplementary App. D. In general, we find that modeling strongly local information with low-order proximity yields good performance (and is computationally cheapest).

7 Conclusion

We have proposed the first unifying perspective that encompasses both proximity-preserving and structural node embedding methods, clarifying their contested technical relationship [4, 3]. This allows us to learn either kind of node embedding from any node proximity matrix that can be computed on a graph, which arises throughout the field of graph mining. Our three-step framework PhUSION opens up a variety of design choices (we empirically study 35), encompassing existing methods and also producing novel ones. We provide insights into productive design choices for node-level graph mining using either kind of embedding. By aggregating a graph's embeddings, we can derive graph-level features from the node proximities; we show pre-

cisely what information we can capture that is lost by other graph kernels and feature learning methods.

Within PhUSION there is still room to explore more design choices, such as other embedding functions (e.g. nonlinear autoencoders used by a few methods for positional node embedding [17], or trainable characteristic function sampling recently proposed for node and graph embedding [23]). For graph embedding, other designs use successive kernel embedding and the incorporation of node attributes [8]. Furthermore, fast approximate computation of node proximities can allow PhUSION to scale to very large graphs [5, 2].

Acknowledgements

This work is supported by NSF Grant No. IIS 1845491, Army Young Investigator Award No. W9-11NF1810397, and Adobe, Amazon, Facebook, and Google faculty awards. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding parties.

References

[1] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 2018.

[2] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In *KDD*, 2018.

[3] Ryan A. Rossi, Di Jin, Sungchul Kim, Nesreen K. Ahmed, Danai Koutra, and John Boaz Lee. On proximity and structural role-based embeddings in networks: Misconceptions, methods, and applications. *TKDD*, 2020.

[4] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. In *ICLR*, 2020.

[5] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*, 2018.

[6] Sudhanshu Chanpuriya and Cameron Musco. Infinitewalk: Deep network embeddings as laplacian embeddings with a nonlinearity. In *KDD*, 2020.

[7] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Netlsd: hearing the shape of a graph. In *KDD*, 2018.

[8] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *NeurIPS*, 2018.

[9] Mark Heimann, Tara Safavi, and Danai Koutra. Distribution of node embeddings as multiresolution features for graphs. In *ICDM*, 2019.

[10] Junchen Jin, Mark Heimann, Di Jin, and Danai Koutra. Understanding and evaluating structural node embeddings. In *KDD MLG Workshop*, 2020.

[11] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. Regal: Representation learning-based graph alignment. In *CIKM*, 2018.

[12] Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. Fast network embedding enhancement via high order proximity approximation. In *IJCAI*, 2017.

[13] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *SDM*, 2013.

[14] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Polo Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *PKDD*, 2011.

[15] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, 2016.

[16] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, 2015.

[17] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, 2016.

[18] Mark Heimann, Goran Murić, and Emilio Ferrara. Structural node embedding in signed social networks: Finding online misbehavior at multiple scales. In *Complex Networks*, 2020.

[19] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *KDD*, 2017.

[20] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NeurIPS*, 2008.

[21] Giannis Nikolentzos and Michalis Vazirgiannis. Enhancing graph kernels via successive embeddings. In *CIKM*, 2018.

[22] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML GRL Workshop*, 2020.

[23] Benedek Rozemberczki and Rik Sarkar. Characteristic functions on graphs: Birds of a feather,

from statistical descriptors to parametric models.
In *CIKM*, 2020.

A Proofs

A.1 Existing Node Embedding Methods as Special Cases of Eq. (3.2). For all the methods in Theorem 4.1, we list the specific choices of node proximity $\Psi()$, nonlinearity $\sigma()$, and embedding $\zeta()$ functions (as well as whether or not they use multiscale proximity) that make them conform to our framework.

- GraphWave [2]: the node proximity $\Psi()$ computes the graph's heat kernel matrix, the nonlinearity $\sigma()$ is the identity function, and the embedding function $\zeta()$ is characteristic function sampling. The multiscale version of GraphWave is given by Equation 3.2.
- NetMF [5]: the node proximity $\Psi()$ computes the graph's PPMI matrix, the nonlinearity $\sigma()$ is `Log`, and the embedding function $\zeta()$ is SVD.
- InfiniteWalk [6]: the node proximity $\Psi()$ computes the PPMI matrix in the window size limit $T = \infty$, or the Laplacian pseudoinverse $\mathbf{L}^+$ as an approximation of this quantity up to a low-rank correction term. The nonlinearity $\sigma()$ is `Log` (or, for the Laplacian pseudoinverse, the authors consider `Bin-p`), and the embedding function $\zeta()$ is SVD.
- HOPE [15]: the node proximity $\Psi()$ computes the personalized pagerank matrix or the common neighbors matrix $\mathbf{A}^2$, the nonlinearity $\sigma()$ is the identity function, and the embedding is SVD (possibly approximated for scalability [15]).
- GraRep [16]: the node proximity $\Psi()$ is derived from powers of the adjacency matrix, the nonlinearity $\sigma()$ is `Log`, and the embedding function is SVD; this method computes multiscale node embeddings by concatenating embeddings derived from different powers of the adjacency matrix.
- DNGR [17]: the node proximity $\Psi()$ computes the graph's PPMI matrix (in a slightly different way than NetMF), the nonlinearity $\sigma()$ is `Log`, and the nonlinear embedding function $\zeta()$ is implemented with a stacked denoising autoencoder.
- sRDE [18]: the node proximity $\Psi()$ in a *signed* network is computed using a signed random walk with restart procedure, the nonlinearity $\sigma()$ is the identity function, and the embedding function $\zeta()$ consists of computing a histogram (which is also permutation-invariant) of each node's signed proximity scores.

A.2 Embedding Functions that Produce Positional vs. Structural Node Embeddings Here we give the proof of Theorem 4.2:

Proof. **Part 1: SVD yields different embeddings for automorphic nodes.** Recall that finding the SVD of $\tilde{\mathbf{S}} = [\tilde{\mathbf{S}}_1, \mathbf{0}; \mathbf{0}, \tilde{\mathbf{S}}_2]$ is equivalent to finding the eigendecomposition of $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top$: the singular vectors (columns of $\mathbf{U}$) are the eigenvectors and the singular values (diagonal entries of $\mathbf{\Sigma}$) the square roots of eigenvalues of $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top$. Since the embeddings are formed from the first d columns of $\mathbf{U}$ and $\mathbf{\Sigma}$, we equivalently analyze the eigendecomposition of $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top$.

1. $\tilde{\mathbf{S}}_1$ and $\tilde{\mathbf{S}}_2$ are similar matrices and thus have the same eigenvalues and eigenvectors.

2. $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top$ has the same eigenvalues as $\tilde{\mathbf{S}}_1$ (equivalently, $\tilde{\mathbf{S}}_2$). First, we show that all eigenvalues of $\tilde{\mathbf{S}}_1, \tilde{\mathbf{S}}_2$ are eigenvalues of $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top$: if $\tilde{\mathbf{S}}_1\mathbf{v}_\lambda = \lambda\mathbf{v}_\lambda$, then $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top[\mathbf{v}_\lambda, \mathbf{0}] = \lambda[\mathbf{v}_\lambda, \mathbf{0}]$; $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top[\mathbf{0}, \mathbf{v}_\lambda] = \lambda[\mathbf{0}, \mathbf{v}_\lambda]$. Conversely, we also show that all eigenvalues of $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top$ are eigenvalues of $\tilde{\mathbf{S}}_1, \tilde{\mathbf{S}}_2$. Without loss of generality we can write any eigenvector $\mathbf{v}$ of $\tilde{\mathbf{S}}$ split in half as $[\mathbf{v}_1, \mathbf{v}_2]$, such that $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top[\mathbf{v}_1, \mathbf{v}_2] = \lambda[\mathbf{v}_1, \mathbf{v}_2]$. Then $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top[\mathbf{v}_1, \mathbf{v}_2] = [\tilde{\mathbf{S}}_1\tilde{\mathbf{S}}_1^\top, \mathbf{0}; \mathbf{0}, \tilde{\mathbf{S}}_2\tilde{\mathbf{S}}_2^\top][\mathbf{v}_1, \mathbf{v}_2] = [\tilde{\mathbf{S}}_1\mathbf{v}_1 + \mathbf{0}\mathbf{v}_2, \mathbf{0}\mathbf{v}_1 + \tilde{\mathbf{S}}_2\mathbf{v}_2] = [\tilde{\mathbf{S}}_1\mathbf{v}_1, \tilde{\mathbf{S}}_2\mathbf{v}_2]$. Since $[\mathbf{v}_1, \mathbf{v}_2]$ was an eigenvector of $\tilde{\mathbf{S}}\tilde{\mathbf{S}}^\top$, $[\tilde{\mathbf{S}}_1\mathbf{v}_1, \tilde{\mathbf{S}}_2\mathbf{v}_2] = \lambda[\mathbf{v}_1, \mathbf{v}_2]$ and thus $\tilde{\mathbf{S}}_1\mathbf{v}_1 = \lambda\mathbf{v}_1$ and $\tilde{\mathbf{S}}_2 = \mathbf{v}_2$, meaning that λ is also an eigenvalue of $\tilde{\mathbf{S}}_1$ and $\tilde{\mathbf{S}}_2$.

3. Thus, each of the top singular vectors of $\tilde{\mathbf{S}}$ that form the dimensions of $\mathbf{Y}$ which form the embedding dimensions up to weighing by the singular values, has the form $[\mathbf{0}, \mathbf{v}_\lambda]$ or $[\mathbf{v}_\lambda, \mathbf{0}]$. (Since the graphs are connected i.e. nonempty, $\mathbf{v}_\lambda \neq \mathbf{0}$.) That is, along any dimension the nodes in one graph will have a nonzero embedding value and the nodes in the other graph will have a zero embedding value.

This is of course an extreme case for a highly contrived example (perfectly automorphic nodes in perfectly disconnected components of a graph), but in general we can see (and the research community has found experimentally on real-world networks) that the SVD embeddings encode positional rather than structural information, and nodes in very different parts of the graph will generally not be close in the embedding space.

Part 2: Permutation-invariant row functions such as CFS yield identical embeddings for automorphic nodes. Let n be the number of nodes in either graph G_1 or G_2. Then the first n nodes in G correspond to G_1 and the second n nodes in graph correspond to G_2. So for node $i \in [1, \ldots, n]$, the ID of its counterpart under the isomorphism π is $\pi(i)+n$. Thus, we want to show that the rows of node i and node $\pi(i) + n$ in $\tilde{\mathbf{S}}$ are equivalent up to permutation. Formally, we show that for any $i, j \in [1, \ldots, n]$, $\tilde{\mathbf{S}}_{ij} = \tilde{\mathbf{S}}_{\pi(i)+n, \pi(j)+n}$.

Let $\mathbf{e}_i$ be the i-th standard basis. Then $\tilde{\mathbf{S}}_{ij} = \tilde{\mathbf{S}}_{1_{ij}} = \mathbf{e}_i\tilde{\mathbf{S}}_1\mathbf{e}_j^\top = \mathbf{e}_i\mathbf{P}^\top\tilde{\mathbf{S}}_2\mathbf{P}\mathbf{e}_j^\top = \mathbf{e}_{\pi(i)}\tilde{\mathbf{S}}_2\mathbf{e}_{\pi(j)}^\top = \tilde{\mathbf{S}}_{2_{\pi(i)\pi(j)}} = \tilde{\mathbf{S}}_{\pi(i)+n,\pi(j)+n}$. This shows that any nonzero element in the i-th row of $\tilde{\mathbf{S}}$ (which must occur in the first n elements) has a corresponding element among the second j elements of the $(\pi(i)+n)$-th row. Of course, the second n elements in the i-th row and the first n elements in the $(\pi(i)+n)$-th row of $\tilde{\mathbf{S}}$ are zeros. Thus, these rows have the same elements and are identical up to permutation. ∎

B Node Proximity Hyperparameters

For positional node embeddings: All embeddings have the standard 128 dimensions [5]. We tuned the hyperparameters of the node proximity functions `PPMI`, `PPR`, `HK`, and `FaBP` on the Wikipedia dataset via grid search over the following values:

1. `HK`: we tried scale values of $s \in [0.01, 0.1, 1, 10, 25, 50]$, and find best performance from $s = 0.1$.
2. `PPR`: we tried decay parameter values $\beta \in [0.9, 0.5, 0.1, 0.01]$, and find best performance from $\beta = 0.01$.
3. `PPMI`: we tried window size $T \in [2, 5, 10]$ and found little difference, so we use $T = 10$ with the approximate NetMF method [5].
4. `FaBP`: we tried values for the parameters $a, c \in \{0.01, 0.1, 1, 10\}$. We found little difference for values of a, but smaller c can lead to better performance, so we chose $a = 1$ and $c = 0.01$.

For structural embeddings: On these smaller graphs, all embeddings are 50-dimensional. We tuned the hyperparameters of the node proximity functions `PPMI`, `PPR`, `HK`, and `FaBP` on the USA dataset via grid search over the following values:

1. `HK`: we used multiscale embeddings following [2]. We found that on the airports datasets, their automatic scale selection procedure yielded unintuitively large and poorly performing scales. [1] Thus, we tried $\{1, 5, 10, 25, 50\}, \{0.1, 1, 10, 25, 50\}$ and $\{0.01, 0.1, 1, 10, 100\}$, and find best performance from the latter.
2. `PPR`: we tried decay parameter values $\beta \in [0.9, 0.5, 0.1, 0.01]$, and find $\beta = 0.01$ works best.
3. `PPMI`: we tried window size $T \in [2, 5, 10]$ and found that $T = 10$ achieves best performance.
4. `FaBP`: we tried parameter values $a, c \in [0.01, 0.1, 1, 10]$, but in the end we found that the heuristic proposed in [14] for setting a and c works best: $a = 4h_h^2/(1-4h_h^2), c = 2h_h/(1-4h_h^2)$. Here the "about-half" homophily factor $h_h = \sqrt{\frac{-c_1+\sqrt{c_1^2+4c_2}}{8c_2}}$ where $c_1 = \mathrm{Tr}(\mathbf{D}) + 2, c_2 = \mathrm{Tr}(\mathbf{D}^2) - 1$.

For graph classification: For `HK`, we use scale parameters $s \in \{0.01, 0.1, 1, 10, 100\}$ to parallel NetLSD. For proximity functions computed by matrix powers (`Adj` and `RW`), we consider powers $k \in \{1, 2, 3, 4, 5\}$. At each of the five parameter settings, we learn 10-dimensional embeddings and use Eq. 3.2 to form a multiscale embedding with 50 dimensions (to match or stay below the modeling capacity of NetLSD and RetGK). Between NetLSD's higher (250) dimension and RetGK's successive kernel embeddings, our experimental setup gives NetLSD and RetGK each a small advantage.

C Clustering Structural Node Embedding: Additional Details and Results

We use the synthetic graph generation pipeline provided by GraphWave [2]. The graphs are given by 5 basic shapes of one of different types ("house", "fan", "star") [2] that are placed on a cycle of length 30. In the main paper, we add 10% random edges to perturb the otherwise perfect role equivalences of nodes in the same part of different shape; however, in Tab. 6, we include clustering results on noiseless networks exhibiting perfect role equivalence. We use agglomerative clustering (with single linkage) to cluster the node embeddings learned by each method.

	`PPMI`	`FaBP`	`HK`	`PPR`	$\mathbf{A}^2$	$\mathbf{R}^2$	$\mathbf{L}^+$
Homogeneity	0.8738	**1.000**	0.9727	**1.000**	**1.000**	0.9297	0.8370
Completeness	0.8367	**1.000**	0.9407	**1.000**	**1.000**	0.9057	0.7812
Silhouette	0.8241	0.9089	0.8814	**0.9702**	0.9204	0.8616	0.8313

Table 6: Clustering results on *noiseless* synthetic datasets. `HK` used in GraphWave is outperformed on all metrics by other proximity matrices.

D Proximity Order in Structural Embedding

The order of proximity that node embeddings model has been shown to be very important. While some methods by default model low-order (e.g. 2nd-order) proximities [10], other methods try to balance low-order and high-order proximities to capture local and global information. This has been done with multiscale embeddings, whether positional [16] or structural [2]. Setting

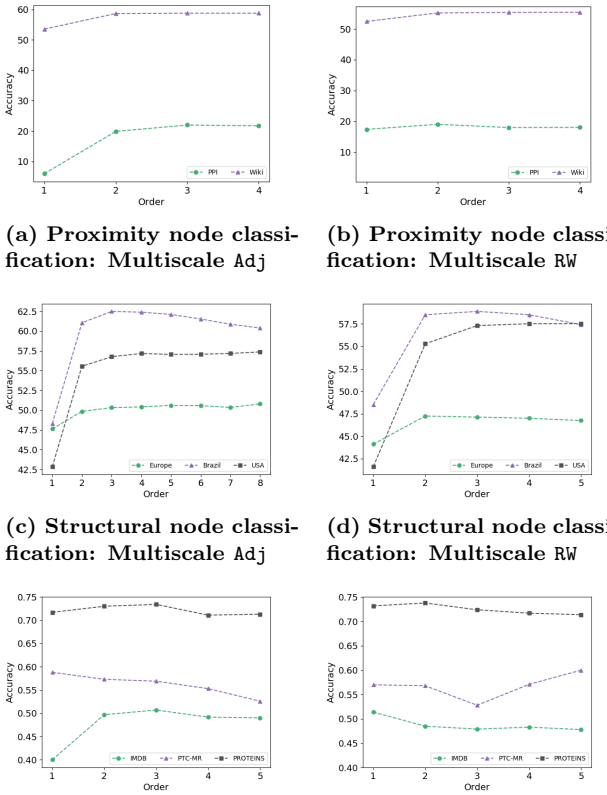
[1] For example, applying the official implemention of GraphWave [2] using the automatic scale selection on USA-airports dataset gives a range of scale parameters $s_{min} = 2014340.3$ and $s_{max} = 8763076.3$.

hyperparameters that govern the order of proximity is thus important to understand.

Setup. For node and graph classification, we consider the effect of varying the order for methods consisting of powers of a (filtered) similarity matrix $\tilde{\mathbf{S}}$ (i.e. `Adj` or `RW`) when computing multiscale embeddings (Eq. 3.2). We only consider up to 4th order for positional node embeddings due to the larger size of those datasets (which is also why we omit the largest dataset BlogCatalog). For any value of k, we compute the embeddings from each power of $\tilde{\mathbf{S}}, \tilde{\mathbf{S}}^2, \ldots, \tilde{\mathbf{S}}^k$ (using `Log` nonlinearity for positional node embeddings and `Identity` for structural embeddings, nonlinearity functions which performed well on average for each kind of embedding in § 6) and concatenate the resulting embeddings. Note that for graph classification experiments, we now learn a 50-dimensional embedding at *each* scale, as we are not comparing to baseline methods now.

Results. The results are shown in Fig. 3. We can see, confirming the intuition of prior structural embedding methods [10] that lower order proximity is sufficient for best performance and saves the computational expense of computing higher order node proximities (which amounts to additional multiplications of increasingly dense matrices).

Observation 5. *Modeling low-order node proximity (however, beyond first-order proximity, or direct edge connections alone) is generally sufficient for both kinds of embedding methods.*



(a) Proximity node classification: Multiscale `Adj`

(b) Proximity node classification: Multiscale `RW`

(c) Structural node classification: Multiscale `Adj`

(d) Structural node classification: Multiscale `RW`

(e) Graph classification: Multiscale `Adj`

(f) Graph classification: Multiscale `RW`

Figure 3: Effect of proximity order on node and graph classification. Low order proximities are sufficient to achieve good performance.

E Proximity Matrix Properties for Effective Node Embeddings

A powerful tool for the design of future node embedding methods would be an *intrinsic* characterization of successful design choices for node embedding; this could allow for effective model selection without relying on *extrinsic* evaluation (i.e. performance on downstream tasks as in § 6). The node embedding step usually leverages standard dimensionality reduction techniques; from a graph mining perspective, the most interesting part is the construction of (potentially nonlinearly transformed) node proximities. Thus, we seek to understand: how can we characterize choices $\Psi(\sigma(\mathbf{A}))$ that yield useful (positional or structural) node embeddings? While effective intrinsic analysis of node embedding methods is a major open question, we present some initial exploratory analysis to prompt further investigation.

E.1 Positional Embeddings In a node proximity matrix, the sums of each row correspond to the total proximity scores each node has to all other nodes. Our intuition is that if we are to expect good positional node embeddings, most nodes should have a moderate amount of total proximity to other nodes—too low, and the embedding objective will have too little similarity information to learn an effective embedding; too high, and the embedding objective will try to embed this node indiscriminately similarly to many other nodes.

Setup. In Fig. 4, we visualize the distribution of row sums of all node proximity matrices, arising from each combination of node proximity and nonlinearity function that we evaluated in this work.

Results. Some of these distributions exhibit a bell curve shape with the values concentrated in the middle of the distribution, while others exhibit a power law distribution with a single long tail. (Note that for the `Bin-5` nonlinearity, the tail is on the left as most values in the matrix are 1, so low row sums are the exception. In general, the tail consists of the large row sums, as is typical for most power law distributions.)

Many successful design choices produce a bell shape

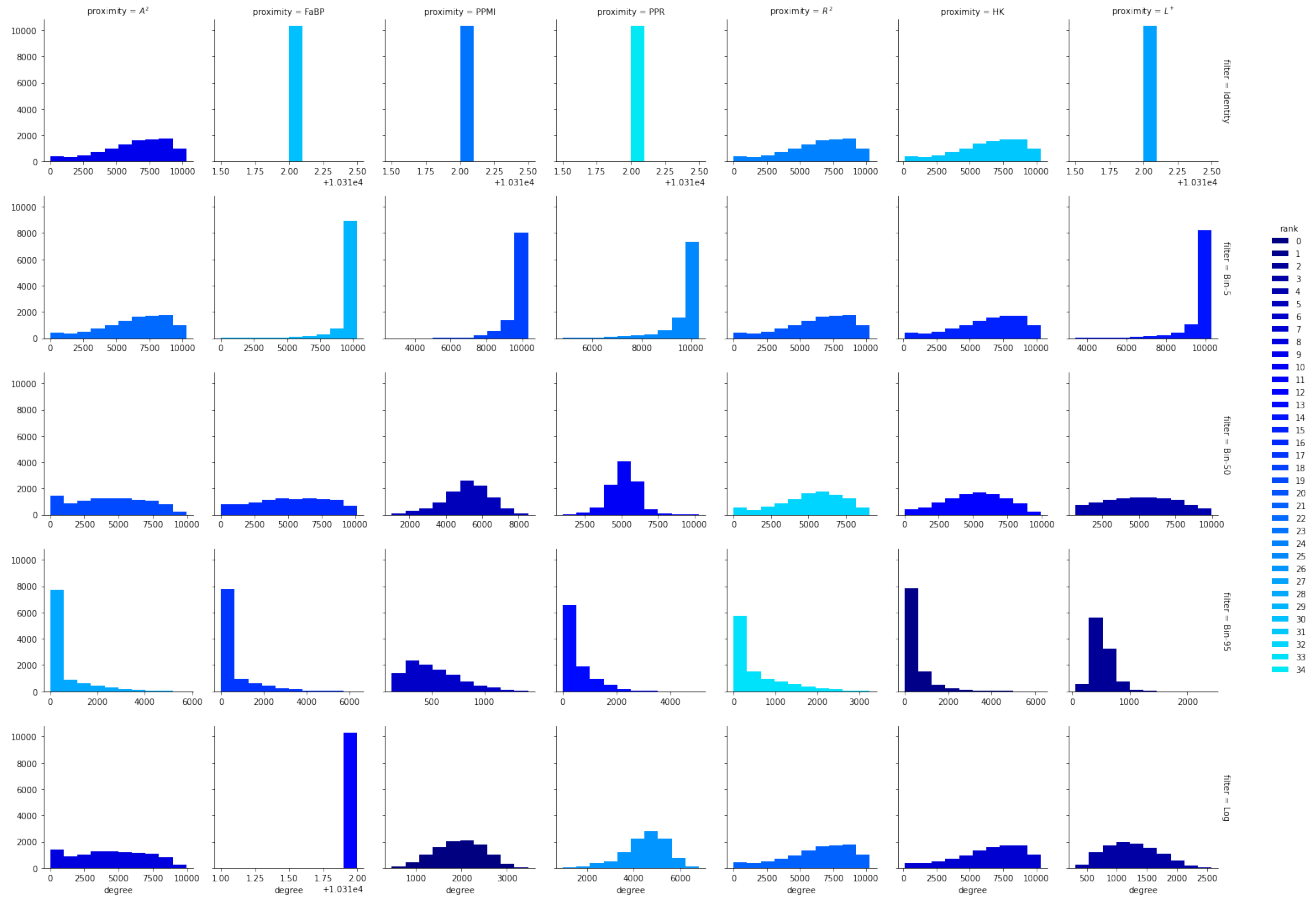


Figure 4: Degree (row sum) distributions of matrices resulting from all different combinations of $\Psi()$ and $\sigma()$ on BlogCatalog. In general, some of the best-performing embedding methods (darker colors) come from matrices whose row sum distributions follow a bell curve rather than a power law.

distribution of row sums. For example, the `Log` nonlinearity filter (the best-performing nonlinearity on average) produces bell-shaped row sum distributions for all proximity matrices. `Bin-95` produces a somewhat bell-shaped distribution for `PPMI`, one of the best-performing proximity matrices. In general, this lines up with our intuition to expect mostly moderate row sums.

Observation 6. *Some of the matrices yielding the best positional node embeddings have a bell curve rather than a power law distribution of row sums: that is, most nodes have moderate total proximity scores to all other nodes.*

E.2 Structural Embeddings The CFS embedding method treats each row of the proximity matrix as a probability distribution. When learning structural embeddings using CFS, GraphWave notes that two cases will be uninformative for structural embedding: if the distribution of row entries is either too uniform or if it has too few nonzero values.

Setup. To measure the row-wise uniformity of the matrices, we consider the variance of each row. Meanwhile, we use entropy to diagnose rows with a few large entries and otherwise mostly small ones. Thus, we plot the row-wise distribution of variances and entropy for each proximity matrix, as well as the distribution of row sums as in § E.1. For brevity, we do not consider nonlinearity. Note that for `PPR` and $\mathbf{L}^+$, we truncate entropy values close to $-\infty$ to -10 for ease of visualization.

Results. For most proximity matrices, the row-wise distribution of all three statistics tends to follow a power law distribution. The row-wise sum and variance distributions for the `PPMI` matrix, which generally leads to some of the the weaker structural embedding methods, tend to follow this pattern much more noisily. Proximity matrices `PPR` and `FaBP`, on the other hand, tend to follow an extreme power law distribution with a very thin tail. This may indicate that a moderate power law distri-

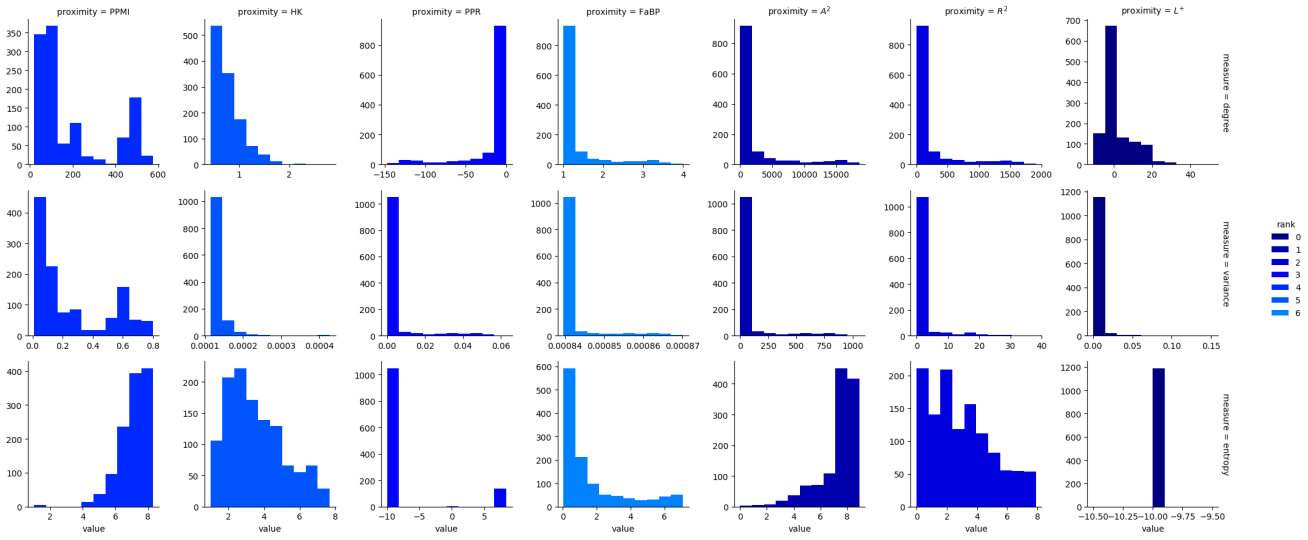


Figure 5: Distribution of row distribution statistics (degree, variance, and entropy) of proximity matrices (without nonlinearity) on USA Airports dataset used for structural embeddings. Methods leading to more accurate structural embeddings have darker color. Some of the best embeddings come from matrices whose rows' variance and entropy follow a power law distribution.

bution may be the most informative for structural embeddings. The contrast with proximity-preserving embeddings (§ E.1), where the most successful embeddings tended to come from matrices with a bell-curve distribution of row sums, corroborates our finding that the best positional and structural node embedding methods tended to use very different design choices.

OBSERVATION 7. *Proximity matrices that yield good structural embeddings often follow a power law distribution of row-wise statistics. The contrast with Obs. 6 indicates that proximity matrices that lead to successful positional or structural node embeddings may have fundamentally different properties.*