A Structural Graph Representation Learning Framework

Ryan A. Rossi
Adobe Research

Nesreen K. Ahmed
Intel Labs

Eunyee Koh
Adobe Research

Sungchul Kim
Adobe Research

Anup Rao
Adobe Research

Yasin Abbasi-Yadkori
VinAI

ABSTRACT

The success of many graph-based machine learning tasks highly depends on an appropriate representation learned from the graph data. Most work has focused on learning node embeddings that preserve proximity as opposed to *structural role-based embeddings* that preserve the structural similarity among nodes. These methods fail to capture higher-order structural dependencies and connectivity patterns that are crucial for structural role-based applications such as visitor stitching from web logs. In this work, we formulate *higher-order network representation learning* and describe a general framework called HONE for learning such structural node embeddings from networks via the subgraph patterns (network motifs, graphlet orbits/positions) in a nodes neighborhood. A general diffusion mechanism is introduced in HONE along with a space-efficient approach that avoids explicit construction of the k-step motif-based matrices using a k-step linear operator. Furthermore, HONE is shown to be fast and efficient with a worst-case time complexity that is nearly-linear in the number of edges. The experiments demonstrate the effectiveness of HONE for a number of important tasks including link prediction and visitor stitching from large web log data.

KEYWORDS

Structural node embeddings, role-based embeddings, structural similarity, roles, network motifs, graphlets, structural embeddings

ACM Reference Format:
Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, Sungchul Kim, Anup Rao, and Yasin Abbasi-Yadkori. 2020. A Structural Graph Representation Learning Framework. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371843

1 INTRODUCTION

Structural role discovery [40] aims to reveal nodes with topologically similar neighborhoods while being possibly far away in the graph or even in different graphs altogether. Intuitively, two nodes belong to the same role if they are structurally similar (with respect to the general connectivity and subgraph patterns in a nodes neighborhood). Roles may represent higher-order subgraph patterns (network motifs) such as star-center (hub) nodes, star-edge nodes, near-cliques or bridge nodes connecting different regions of the graph. Most work on embeddings have focused on preserving the notion of proximity (closeness) as opposed to the notion of structural similarity proposed in [40]. As such, two nodes with similar proximity-based embeddings are guaranteed to be near one another in the graph (a property of communities [17]). However, learning structural role-based embeddings that preserve the notion of structural similarity are important for many predictive modeling applications [41] such as the visitor stitching task where the goal is to predict the web sessions that belong to the same user.

To address this problem and learn more appropriate *structural role-based embeddings* for such applications, we propose a general framework called Higher-Order Network Embeddings (HONE) for learning higher-order structural node embeddings based on network motifs (graphlets). The approach leverages all available motif counts by deriving a weighted motif graph $\mathbf{W}_t$ from each network motif $H_t \in \mathcal{H}$ and uses these as a basis to learn higher-order structural node embeddings. The HONE framework expresses a new class of structural node embedding methods based on a set of motif-based matrices and their powers. We also introduce diffusion-based HONE variants that leverage a general diffusion mechanism to improve predictive performance. Furthermore, this work describes a space-efficient approach to avoid explicit construction of the k-step motif-based matrices by defining a k-step linear operator. The time complexity of HONE is shown to be linear in the number of edges and therefore is fast and scalable for large networks. Empirically, we investigate the HONE variants and their properties extensively in Section 4. The experiments demonstrate the effectiveness of *higher-order structural embeddings* for link prediction as we achieve a mean relative gain in AUC of 19% over all embedding methods and network data sets. In addition, the diffusion-based HONE variants achieve a mean gain of 1.97% in AUC over the other HONE variants. We also demonstrate the effectiveness of HONE for visitor stitching using two real-world company data sets with known ground-truth. Finally, HONE is shown to capture roles (structural similarity) as it successfully uncovers the actual exact role assignments in graphs with known ground-truth.

Contributions: This work makes three important contributions. First, we introduce the problem of higher-order (motif-based) network embedding. Second, we propose a general class of methods for learning such structural (role-based) embeddings via higher-order network motifs. The resulting embeddings are shown to be role-based (structural) and capture the notion of structural similarity (roles) [40] as opposed to proximity/community-based embeddings

WSDM '20, February 3–7, 2020, Houston, TX, USA

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371843

that have been the focus of most previous work. Third, we demonstrate the effectiveness of learning *structural higher-order network embeddings* for link prediction and visitor stitching of web logs.

2 HIGHER-ORDER NETWORK EMBEDDINGS

This section proposes a new class of embedding models called *Higher-Order Network Embeddings* (HONE) and a general framework for deriving them. The class of higher-order network embedding methods is defined as follows:

Definition 1 (Higher-Order Network Embeddings). *Given a network $G = (V, E)$, a set of network motifs $\mathcal{H} = \{H_1, \ldots, H_T\}$, the goal of higher-order network embedding (HONE) is to learn a function $f : V \rightarrow \mathbb{R}^D$ that maps nodes to D-dimensional* structural node embeddings *using network motifs $\mathcal{H}$.*[1]

The particular family of higher-order structural node embeddings presented in this work are based on learning a function $f : V \rightarrow \mathbb{R}^D$ that maps nodes to D-dimensional embeddings using (powers of) weighted motif graphs derived from a structural motif matrix function Ψ. However, many other families of *higher-order structural node embedding* methods exist in the class of higher-order network embeddings (Definition 1).

We summarize the main steps of the HONE framework in Algorithm 1 (with the exception of attribute diffusion discussed in Section 2.6 and normalization, which are both optional). For clarity, Algorithm 1 also summarizes the organization and shows the connections between Sections 2.1-2.5.

2.1 Network Motifs

The HONE framework can use graphlets or orbits; and both can be computed fast in only a few seconds for very large networks, see [1, 4]. Recall that the term network motif is used generally in this work and may refer to graphlets or orbits (graphlet automorphisms) [1, 35]. A graphlet $H_t = (V_k, E_k)$ is an induced subgraph consisting of a subset $V_k \subset V$ of k vertices from $G = (V, E)$ together with all edges whose endpoints are both in this subset $E_k = \{\forall e \in E \mid e = (u, v) \wedge u, v \in V_k\}$. Alternatively, the nodes of every graphlet can be partitioned into a set of automorphism groups called orbits [35]. It is important to consider the *position* of an edge in a graphlet, for instance, an edge in the 4-node path (Figure 1) has two different unique positions, namely, an edge on the outside of the 4-node path (H_4 in Figure 1) or the edge in the center of the path (H_5). Each unique edge position in a graphlet is called an orbit. In this work, we use all (2-4)-vertex connected edge orbits and denote this set as $\mathcal{H}$ (Figure 1).

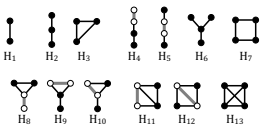


Figure 1: All (2-4)-vertex connected edge orbits. For graphlets with more than one edge orbit (*e.g.*, 4-path-edge orbit H_4 and 4-path-center orbit H_5), the gray edge (between the unshaded nodes) is used to distinguish between the different edge orbits of the graphlet.

[1]Network motifs is used generally to refer to either induced subgraphs/graphlets or graphlet orbits (Section 2.1).

Algorithm 1 Higher-Order Network Embedding (HONE)

Step 1: Given a network $G = (V, E)$ with $N = |V|$ nodes and a set $\mathcal{H} = \{H_1, \ldots, H_T\}$ of T motifs (Section 2.1), form the weighted motif adjacency matrices $\mathcal{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_T\}$ where $(\mathbf{W}_t)_{ij}$ = # of instances of motif $H_t \in \mathcal{H}$ between node i and j (Section 2.2).

Step 2: Derive all k-step motif matrices for all T motifs and K steps:

$$\mathbf{S}_t^{(k)} = \Psi(\mathbf{W}_t^k), \quad \text{for } k = 1, \ldots, K \text{ and } t = 1, \ldots, T$$

where Ψ is a motif matrix function from Section 2.3.

Step 3: Find low-rank "local" structural node embeddings for each k-step motif matrix $\mathbf{S}_t^{(k)}$ by solving Eq. 12 (Section 2.4).

Step 4: Concatenate all low-dimensional structural node embeddings for all T network motifs and K steps to obtain $\mathbf{Y}$ (Eq. 15).

Step 5: Given $\mathbf{Y}$, find a "global" low-dimensional rank-D structural node embedding matrix $\mathbf{Z}$ by solving Eq. 17 (Section 2.5) and return $\mathbf{Z} \in \mathbb{R}^{N \times D}$.

2.2 Weighted Motif Graphs

Given a network $G = (V, E)$ with $N = |V|$ nodes, $M = |E|$ edges, and a set $\mathcal{H} = \{H_1, \ldots, H_T\}$ of T network motifs, we form the weighted motif adjacency matrices: $\mathcal{W} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_T\}$ where

$$(\mathbf{W}_t)_{ij} = \#\text{ occurrences of motif } H_t \in \mathcal{H} \text{ that contain } (i, j) \in E$$

The weighted motif graphs differ from the original graph in two important and fundamental ways. First, the edges in each motif graph is likely to be weighted differently. This is straightforward to see as each network motif can appear at a different frequency than another arbitrary motif for a given edge. Intuitively, the edge motif weights when combined with the structure of the graph reveal important structural properties with respect to the weighted motif graph. Second, the motif graphs are often *structurally* different as shown in Figure 2. For instance, if edge $(i, j) \in E$ exists in the original graph G, but $(\mathbf{W}_t)_{ij} = 0$ for some arbitrary motif H_t, then $(i, j) \notin E_t$ where E_t is the edge set for motif $H_t \in \mathcal{H}$. Hence, the motif graphs encode relationships between nodes that have a sufficient number of motifs. To generalize the above weighted motif graph formulation, we replace the edge constraint that says an edge exists between i and j if the number of instances of motif $H_t \in \mathcal{H}$ that contain nodes i and j is 1 or larger, by enforcing an edge constraint that requires each edge to have at least δ motifs. In other words, different motif graphs can arise using the same motif H_t by enforcing an edge constraint that requires each edge to have at least δ motifs. This is an important property of the above formulation.

2.3 Structural Motif Matrix Functions

To generalize HONE for any *motif-based matrix formulation*, we define Ψ as a function $\Psi : \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times N}$ over a weighted motif adjacency matrix $\mathbf{W}_t \in \mathcal{W}$. Using Ψ we derive

$$\mathbf{S}_t = \Psi(\mathbf{W}_t), \quad \text{for } t = 1, 2, \ldots, T \tag{1}$$

The term *motif-based matrix* refers to any motif matrix $\mathbf{S}$ derived from $\Psi(\mathbf{W})$.[2] We summarize a few motif matrix functions Ψ below.

- **Weighted Motif Graph**: Given a graph G and a network motif $H_t \in \mathcal{H}$, form $\mathbf{W}_t$ where H_t : $(\mathbf{W}_t)_{ij}$ = number of instances of

[2]For convenience, $\mathbf{W}$ denotes a weighted adjacency matrix for an arbitrary motif.

H_t that contain nodes i and j. In the case of using HONE directly with a weighted motif adjacency matrix $\mathbf{W}$, then

$$\Psi : \mathbf{W} \rightarrow \mathbf{IW} \tag{2}$$

The number of paths weighted by motif counts from node i to node j in k-steps is given by

$$(\mathbf{W}^k)_{ij} \;=\; \big(\underbrace{\mathbf{W} \cdots \mathbf{W}}_{k} \big)_{ij} \tag{3}$$

- **Weighted Motif Transition Matrix**: The random walk on a graph $\mathbf{W}$ weighted by motif counts has transition probabilities

$$P_{ij} = \frac{W_{ij}}{w_i} \tag{4}$$

where $w_i = \sum_j W_{ij}$ is the motif degree of node i. The random walk motif transition matrix $\mathbf{P}$ for an arbitrary weighted motif graph $\mathbf{W}$ is defined as:

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{W} \tag{5}$$

where $\mathbf{D} = \texttt{diag}(\mathbf{We}) = \texttt{diag}(w_1, w_2, \ldots, w_N)$ is a $N \times N$ diagonal matrix with the motif degree $w_i = \sum_j W_{ij}$ of each node on the diagonal called the *diagonal motif degree matrix* and $\mathbf{e} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$ is the vector of all ones. $\mathbf{P}$ is a row-stochastic matrix with $\sum_j P_{ij} = \mathbf{p}_i^T \mathbf{e} = 1$ where $\mathbf{p}_i \in \mathbb{R}^N$ is a column vector corresponding to the i-th row of $\mathbf{P}$. For directed graphs, the motif out-degree is used. However, one can also leverage the motif in-degree or total motif degree (among other quantities). The motif transition matrix $\mathbf{P}$ represents the transition probabilities of a non-uniform random walk on the graph that selects subsequent nodes with probability proportional to the connecting edge's motif count. Therefore, the probability of transitioning from node i to node j depends on the motif degree of j relative to the total sum of motif degrees of all neighbors of i. The probability of transitioning from node i to j in k-steps is given by

$$(\mathbf{P}^k)_{ij} \;=\; \big(\underbrace{\mathbf{P} \cdots \mathbf{P}}_{k} \big)_{ij} \tag{6}$$

- **Weighted Motif Laplacian**: The motif Laplacian for a weighted *motif* graph $\mathbf{W}$ is defined as:

$$\mathbf{L} \;=\; \mathbf{D} - \mathbf{W} \tag{7}$$

where $\mathbf{D} = \texttt{diag}(\mathbf{We})$ is the diagonal motif degree matrix defined as $D_{ii} = \sum_j W_{ij}$. For directed graphs, we can use either in-motif degree or out-motif degree.

- **Normalized Weighted *Motif* Laplacian**: Given a graph $\mathbf{W}$ weighted by the counts of an arbitrary network motif $H_t \in \mathcal{H}$, the *normalized motif Laplacian* is defined as

$$\widehat{\mathbf{L}} \;=\; \mathbf{I} \;-\; \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2} \tag{8}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{D} = \texttt{diag}(\mathbf{We})$ is the $N \times N$ diagonal matrix of motif degrees.

- **Random Walk Normalized Weighted Motif Laplacian**: Formally, the *random walk normalized motif Laplacian* is

$$\widehat{\mathbf{L}}_{\text{rw}} \;=\; \mathbf{I} \;-\; \mathbf{D}^{-1}\mathbf{W} \tag{9}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{D}$ is the motif degree diagonal matrix with $D_{ii} = w_i, \forall i = 1, \ldots, N$, and $\mathbf{W}$ is the weighted motif adjacency matrix for an arbitrary motif $H_t \in \mathcal{H}$. Observe that $\widehat{\mathbf{L}}_{\text{rw}} = \mathbf{I} - \mathbf{P}$ where $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$ is the motif transition matrix of a random walker on the weighted motif graph.

Notice that all variants are easily formulated as functions Ψ in terms of an arbitrary motif weighted graph $\mathbf{W}$.

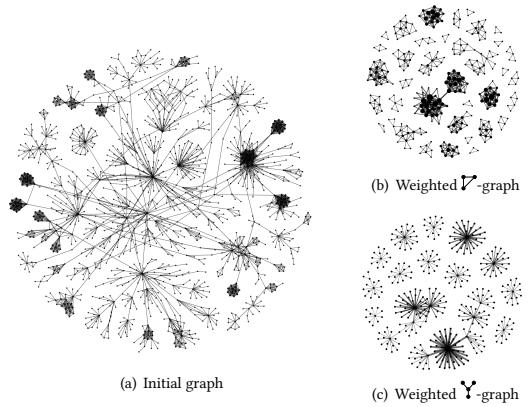


Figure 2: Motif graphs differ in structure *and* weight. Size (weight) of nodes and edges in the triangle and 4-star graphs correspond to the frequency of triangles and 4-stars.

2.4 K-Step Motif-based Structural Embeddings

We describe the *local* higher-order structural node embeddings learned for each network motif $H_t \in \mathcal{H}$ and k-step where $k \in \{1, \ldots, K\}$. The term local refers to the fact that structural node embeddings are learned for each individual motif and k-step independently. We define k-step motif-based matrices for all T motifs and K steps as follows:

$$\mathbf{S}_t^{(k)} = \Psi(\mathbf{W}_t^k), \quad \text{for } k = 1, \ldots, K \text{ and } t = 1, \ldots, T \tag{10}$$

where

$$\Psi(\mathbf{W}_t^k) \;=\; \Psi(\underbrace{\mathbf{W}_t \;\cdots\; \mathbf{W}_t}_{k}) \tag{11}$$

These k-step motif-based matrices can densify quickly and therefore the space required to store the k-step motif-based matrices can grow fast as K increases. For large graphs, it is often impractical to store the k-step motif-based matrices for any reasonable K. To overcome this issue, we avoid *explicitly* constructing the k-step motif-based matrices entirely. Hence, no additional space is required and we never need to store the actual k-step motif-based matrices for $k > 1$. We discuss and show this for any k-step motif-based matrix later in this subsection.

Given a k-step motif-based matrix $\mathbf{S}_t^{(k)}$ for an arbitrary network motif $H_t \in \mathcal{H}$, we find an embedding by solving the following optimization problem:

$$\underset{\mathbf{U}_t^{(k)}, \mathbf{V}_t^{(k)} \in C}{\arg\min} \;\; \mathbb{D}\big(\mathbf{S}_t^{(k)} \,\|\, \Phi\langle \mathbf{U}_t^{(k)} \mathbf{V}_t^{(k)} \rangle \big), \;\; \forall k = 1,\ldots,K \text{ and } \; t = 1,\ldots,T \tag{12}$$

where $\mathbb{D}$ is a generalized Bregman divergence (and quantifies $\approx$ in the HONE embedding model $\mathbf{S}_t^{(k)} \approx \Phi\langle \mathbf{U}_t^{(k)} \mathbf{V}_t^{(k)} \rangle$) with matching linear or non-linear function Φ and C is constraints (*e.g.*, non-negativity constraints $\mathbf{U} \geq 0$, $\mathbf{V} \geq 0$, orthogonality constraints $\mathbf{U}^T\mathbf{U} = \mathbf{I}, \mathbf{V}^T\mathbf{V} = \mathbf{I}$). The above optimization problem finds low-rank

embedding matrices $\mathbf{U}_t^{(k)}$ and $\mathbf{V}_t^{(k)}$ such that $\mathbf{S}_t^{(k)} \approx \Phi\langle \mathbf{U}_t^{(k)}\mathbf{V}_t^{(k)}\rangle$. The function Φ allows non-linear relationships between $\mathbf{U}_t^{(k)}\mathbf{V}_t^{(k)}$ and $\mathbf{S}_t^{(k)}$. Different choices of Φ and $\mathbb{D}$ yield different HONE embedding models and depend on the distributional assumptions on $\mathbf{S}_t^{(k)}$. For instance, minimizing squared loss with an identity link function Φ yields singular value decomposition corresponding to a Gaussian error model [18]. Other choices of Φ and $\mathbb{D}$ yield other HONE embedding models with different error models including Poisson, Gamma, or Bernoulli distributions, see [14] for more details.

Recall from above that we avoid *explicitly* computing and storing the k-step motif-based matrices from Eq. 10 as they can densify quickly as K increases and therefore are impractical to store for any large graph and reasonable K. This is accomplished by defining a linear operator corresponding to the K-step motif-based matrices that can run in at most K times the linear operator corresponding to the (1-step) motif-based matrix. In particular, many algorithms used to compute low-rank approximations of large sparse matrices [20, 38] do not need access to the *explicit matrix*, but only the *linear operator* corresponding to action of the input matrix on vectors. For a matrix $\mathbf{A}$, let $T_{\mathbf{A}}$ denote the upper bound on the time required to compute $\mathbf{Ax}$ for any vector $\mathbf{x}$. We note $T_{\mathbf{A}} = O(M)$ where $M = \mathsf{nnz}(\mathbf{A})$ always holds and is a useful bound when $\mathbf{A}$ is sparse. Therefore, the time required to compute a rank-D_ℓ approximation of $\mathbf{A}$ is $O(T_{\mathbf{A}} D_\ell \log N + N D_\ell^2 \log N)$ where $N = |V|$. Note the log term is the number of iterations when Krylov methods are used, see [29] for more details.

Now, we can define a linear operator corresponding to the K-step motif-based matrices that can run in at most K times the linear operator corresponding to the (1-step) motif-based matrix. We show this for the case of any weighted motif adjacency matrix $\mathbf{W}$. Let $T_{\mathbf{W}}$ be the time required to compute $\mathbf{Wx}$, for any vector $\mathbf{x}$. Then, to compute $\mathbf{W}^K\mathbf{x}$, we can do the following. Let $\mathbf{x}_0 \leftarrow \mathbf{x}$ and iteratively compute $\mathbf{x}_i = \mathbf{W}\mathbf{x}_{i-1}$ for $i = 1, \ldots, K$. This shows that $T_{\mathbf{W}^K} = O(KT_{\mathbf{W}})$. This implies that we can compute a rank-D_ℓ embedding of the K-step motif adjacency matrix in time at most $O(KT_{\mathbf{W}} D_\ell \log N + N D_\ell^2 \log N)$ which is at most

$$O(KMD_\ell \log N + N D_\ell^2 \log N) \tag{13}$$

where $M = \mathsf{nnz}(\mathbf{W})$. This implies that the time to compute the rank-D_ℓ embedding grows only linearly with K. Therefore, no additional space is required and we never need to derive/store the actual k-step motif-based matrices for $k > 1$. Moreover, as shown above, the time complexity grows linearly with K and is therefore efficient. The time complexity in Eq. 13 is for singular value decomposition/eigen-decomposition and hence finds the *best* rank-D_ℓ approximation [18]. However, linear operators can also be defined for other optimization techniques that can be used to find a rank-D_ℓ approximation such as stochastic gradient descent, block/cyclic coordinate descent, or alternating least squares. Thus, the time complexity for computing rank-D_ℓ embeddings using these optimization techniques will also only increase by a factor of K.

2.5 Learning *Global* Higher-Order Embeddings

How can we learn higher-order structural node embeddings for an arbitrary graph G that automatically captures the important motifs? Simply concatenating the previous motif embeddings into a single matrix and using this for prediction assumes that each motif is equally important. However, it is obvious that some motifs are more important than others and the choice depends on the graph structure and its properties [1, 35]. Therefore, instead of assuming all motifs contribute equally to the embedding, we learn a *global* higher-order embedding that automatically captures the important motifs in the embedding without requiring an expert to hand select the most important motifs to use. This effectively allows the overall model to learn a combination of latent features using the local motif-based embeddings from different network motifs and from different steps (motif graph scales).

For this, we first normalize the columns of $\mathbf{U}_t^{(k)}$ by a function $g : \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$ as follows:

$$\mathbf{U}_t^{(k)} \leftarrow g(\mathbf{U}_t^{(k)}), \quad \text{for } t = 1, \ldots, T \text{ and } k = 1, \ldots, K \tag{14}$$

In this work, g is a function that normalizes each column of $\mathbf{U}_t^{(k)}$ using the Euclidean norm. Afterwards, we concatenate the k-step node embedding matrices for all T motifs and all K steps:

$$\mathbf{Y} = \Big[\underbrace{\mathbf{U}_1^{(1)} \cdots \mathbf{U}_T^{(1)}}_{\text{1-step}} \cdots \underbrace{\mathbf{U}_1^{(K)} \cdots \mathbf{U}_T^{(K)}}_{K\text{-steps}} \Big] \tag{15}$$

where $\mathbf{Y}$ is a $N \times TKD_\ell$ matrix. Notice that at this stage, we could simply output $\mathbf{Y}$ as the final motif-based node embeddings and use it for a downstream prediction task. However, using $\mathbf{Y}$ directly essentially treats all motifs equally while it is known that some motifs are more important than others and the specific set of important motifs widely depends on the underlying graph structure. Therefore, by learning node embeddings from $\mathbf{Y}$, we can automatically capture the important structure in the data pertaining to certain motifs and avoid trying to specify the important motifs by hand.

Given $\mathbf{Y}$ from Eq. 15, we learn a *global* higher-order network embedding by solving the following:

$$\underset{\mathbf{Z},\mathbf{H}\in C}{\arg\min} \; \mathbb{D}\big(\mathbf{Y} \parallel \Phi\langle \mathbf{ZH}\rangle\big) \tag{16}$$

where $\mathbf{Z}$ is a $N \times D$ matrix of higher-order node embeddings and $\mathbf{H}$ is a $D \times TKD_\ell$ matrix of the latent k-step motif embeddings. Each row of $\mathbf{Z}$ is a D-dimensional embedding of a node. Similarly, each column of $\mathbf{H}$ is an embedding of a latent k-step motif feature (*i.e.*, column of $\mathbf{Y}$) in the same D-dimensional space. In Eq. 16 we use Frobenius norm which leads to the minimization problem:

$$\min_{\mathbf{Z},\mathbf{H}} \; \frac{1}{2}\big\|\mathbf{Y} - \mathbf{ZH}\big\|_F^2 = \frac{1}{2}\sum_{ij}\big(\mathbf{Y}_{ij} - (\mathbf{ZH})_{ij}\big)^2 \tag{17}$$

A similar minimization problem is solved for Eq. 12.

2.6 Attribute Diffusion

Attributes can also be diffused and incorporated into the higher-order structural node embeddings. One approach is to use the motif transition probability matrix as follows:

$$\begin{aligned} \bar{\mathbf{X}}_t^{(0)} &\leftarrow \mathbf{X}, \quad \mathbf{P}_t = \mathbf{D}_t^{-1}\mathbf{W}_t \\ \bar{\mathbf{X}}_t^{(k)} &= \mathbf{P}_t\bar{\mathbf{X}}_t^{(k-1)}, \quad \text{for } k = 1, 2, \ldots, K \end{aligned} \tag{18}$$

where $\mathbf{X}$ is an $N \times F$ attribute matrix and $\bar{\mathbf{X}}_t^{(k)} \in \mathbb{R}^{N\times F}$ is the diffused feature matrix after k-steps. Here $\mathbf{P}_t$ can be replaced by

any of the previous motif-based matrices derived from any motif matrix formulation in Section 2.3. More generally, we define *linear attribute diffusion* for HONE as:

$$\bar{\mathbf{X}}_t^{(0)} \leftarrow \mathbf{X}$$
$$\bar{\mathbf{X}}_t^{(k)} = \Psi(\mathbf{W}_t^{(k)})\bar{\mathbf{X}}_t^{(k-1)}, \quad \text{for } k = 1, 2, \ldots, K \tag{19}$$

More complex attribute diffusion processes can also be formulated such as the *normalized motif Laplacian attribute diffusion*. The resulting diffused attributes are effectively smoothed by the attributes of related nodes governed by the particular diffusion process. Afterwards, we concatenate the diffused attributes for all T motifs $\bar{\mathbf{X}} = \begin{bmatrix} \bar{\mathbf{X}}_1 & \cdots & \bar{\mathbf{X}}_T \end{bmatrix}$ and incorporate them into the node embeddings given as output in Eq. 16 by replacing $\mathbf{Y}$ in Eq. 15 with:

$$\mathbf{Y} = \Big[\underbrace{\mathbf{U}_1^{(1)} \cdots \mathbf{U}_T^{(1)}}_{\text{1-step}} \cdots \underbrace{\mathbf{U}_1^{(K)} \cdots \mathbf{U}_T^{(K)}}_{K\text{-steps}} \bar{\mathbf{X}} \Big] \tag{20}$$

Alternatively, we can concatenate $\bar{\mathbf{X}}$ to $\mathbf{Z}$, $\begin{bmatrix} \mathbf{Z} & \bar{\mathbf{X}} \end{bmatrix}$. The columns of $\bar{\mathbf{X}}$ are normalized using Eq. 14 with the same norm as before.

3 ANALYSIS

Define $\rho(\mathbf{A})$ as the density of $\mathbf{A}$.

Claim 3.1. *Let $\mathbf{W}$ denote an arbitrary k-vertex motif adjacency matrix where $k > 2$, then $\rho(\mathbf{A}) \geq \rho(\mathbf{W})$.*

This is straightforward to see as the motif adjacency matrix constructed from the edge frequency of any motif H with more than $k > 2$ nodes can be viewed as an additional constraint over the initial adjacency matrix $\mathbf{A}$. Therefore, in the extreme case, if every edge contains at least one occurrence of motif H then $\rho(\mathbf{A}) = \rho(\mathbf{W})$. However, if there exists at least one edge that does not contain an instance of H then $\rho(\mathbf{A}) > \rho(\mathbf{W})$. Therefore, $\rho(\mathbf{A}) \geq \rho(\mathbf{W})$.

3.1 Time Complexity

Let $M = |E|$, $N = |V|$, Δ = the maximum degree, T = the number of motifs, K = the number of steps, D_ℓ = number of dimensions for each local motif embedding (Section 2.4), and D = dimensionality of the final node embeddings (Section 2.5).

Lemma 3.1. *The time complexity of HONE is linear in the number of edges M when $M \gg N$, and specifically,*

$$O(M\Delta_{\text{ub}} + MKTD_\ell + NDKTD_\ell) \tag{21}$$

Hence, HONE is fast and efficient for large networks.

Proof. The time complexity of each step is provided below. For the specific HONE embedding model, we assume $\mathbb{D}$ is squared loss, Φ is the identity link function, and no hard constraints are imposed on the objective function in Eq. 12 and Eq. 16.

Network motif frequencies: Fast and efficient algorithms for counting network motifs/graphlets in very large graphs have become common place (*e.g.*, PGD [1] takes a few seconds to count graphlets in very large networks with hundreds of millions of edges). To derive the network motif frequencies, we use recent provably accurate estimation methods [4]. These methods achieve estimates within a guaranteed level of accuracy and time by setting a few simple parameters in the estimation algorithm. The time complexity to estimate the frequency of all network motifs with $\{2, 3, 4\}$-nodes is $O(M\Delta_{\text{ub}})$ in the worst case where Δ_{ub} is a small constant. Hence, Δ_{ub} represents the maximum sampled degree and can be set by the user. See [4] for more details.

Weighted motif graphs: After obtaining the frequencies of the network motifs, we derive a sparse *weighted motif adjacency matrix* for each of the network motifs. The time complexity for each weighted motif adjacency matrix is at most $O(M)$ and this is repeated T times for a total time complexity of $O(MT)$ where T is a small constant. This gives a total time complexity of $O(M(T + \Delta_{\text{ub}}))$ for this step and thus linear in the number of edges.

Structural motif matrix functions: The time complexity of all motif matrix functions Ψ in Section 2.3 is $O(M)$. Since $\Psi(\mathbf{W}_t)$ for $t = 1, \ldots, T$, the total time complexity is $O(MT)$ in the worst case. By Claim 3.1, $M \geq M_t, \forall t$ where $M_t = \texttt{nnz}(\mathbf{W}_t)$ and thus the actual time is likely to be much smaller especially given the rarity of some network motifs in sparse networks such as 4-cliques and 4-cycles.

Embedding each k-step motif graph: For a single weighted motif-based matrix, the time complexity per iteration of cyclic/block coordinate descent [24] and stochastic gradient descent (SGD) [32, 54] is at most $O(MD_\ell)$ where $D_\ell \ll M$. Recall from Section 2.4 that we avoid *explicitly* computing and storing the k-step motif-based matrices by defining a linear operator corresponding to the K-step motif-based matrices with a time complexity that is at most K times the linear operator corresponding to the 1-step motif-based matrix. Therefore, the total time complexity for learning structural node embeddings for all k-step motif-based matrices is:

$$O\Big(\underbrace{TMD_\ell}_{k=1} + \underbrace{2(TMD_\ell)}_{k=2} + \cdots + \underbrace{K(TMD_\ell)}_{k=K} \Big) = O\big(KTMD_\ell\big) \tag{22}$$

Global higher-order structural node embeddings: Afterwards, all k-step motif embedding matrices are horizontally concatenated to obtain $\mathbf{Y}$ (Eq. 15). Each node embedding matrix is $N \times D_\ell$ and there are $K \cdot T$ of them. Thus, it takes $O(NKTD_\ell)$ time to concatenate them to obtain $\mathbf{Y}$. Notice that $N \gg KTD_\ell$ and therefore this step is linear in the number of nodes $N = |V|$. Furthermore, the time complexity for normalizing all columns of $\mathbf{Y}$ is $O(NKTD_\ell)$ for any normalization function g.

Given a *dense* tall-and-skinny matrix $\mathbf{Y}$ of size $N \times KTD_\ell$ where $N \gg KTD_\ell$, we learn the higher-order node embedding matrix $\mathbf{Z}$ and the latent motif embedding matrix $\mathbf{H}$. The time complexity per iteration of cyclic/block coordinate descent [24] and SGD [32, 54] is $O(DNKTD_\ell)$ and thus linear in the number of nodes.

3.2 Space Complexity

Lemma 3.2. *The total space complexity of HONE is*

$$O(T(M + NKD_\ell) + D(N + TKD_\ell)) \tag{23}$$

Proof. The weighted motif adjacency matrices $W_1, \ldots, W_T$ take at most $O(MT)$ space. Similarly, the motif-based matrices derived from any motif matrix function Ψ is at most $O(MT)$. Recall that the space required for some motif-based matrices where the motif being encoded is rare will be much less than $O(MT)$ (Claim 3.1). The space complexity of each k-step motif embedding is $O(ND_\ell)$ and therefore it takes $O(NTKD_\ell)$ space for all $k = 1, \ldots, K$ and $t =$

Table 1: Network statistics (M=million).

Tasks	Graph	$\|V\|$	$\|E\|$	d_{avg}
Visitor Stitching	Comp-A (web logs)	8.9M	55.2M	6.2
	Comp-B (web logs)	22.8M	61.3M	2.7
Link Prediction	soc-hamster	2426	33260	13.7
	rt-twitter-cop	761	2058	2.7
	soc-wiki-Vote	889	5828	6.6
	tech-routers-rf	2113	13264	6.3
	facebook-PU	7315	89733	12.3
	infra-openflights	2939	31354	10.7
	soc-bitcoinA	7604	28248	3.7

$1, \ldots, T$ embedding matrices. Storing the higher-order structural node embedding matrix $\mathbf{Z}$ takes $O(ND)$ space and the k-step motif embedding matrix $\mathbf{H}$ is $O(D \cdot TKD_\ell)$. Therefore, the total space complexity for $\mathbf{Z}$ and $\mathbf{H}$ is $O(ND + DTKD_\ell) = O(D(N + TKD_\ell))$.

4 EXPERIMENTS

This section demonstrates the effectiveness of the proposed framework for a number of important real-world tasks.

4.1 Experimental Setup

We compare the proposed HONE variants to five baseline methods including node2vec [19], DeepWalk [34], LINE [47], GraRep [9], and Spectral clustering [48]. All methods output ($D = 128$)-dimensional node embeddings $\mathbf{Z} = \left[\mathbf{z}_1 \cdots \mathbf{z}_N\right]^T$ where $\mathbf{z}_i \in \mathbb{R}^D$. For LINE, we use 2nd-order-proximity and the number of samples $T = 60$ million [47]. For GraRep, we set $D = 128$ and perform a grid search over $K \in \{1, 2, 3, 4\}$ [9]. For DeepWalk, we use $R = 10$, $L = 80$, and $\omega = 10$ [34]. For node2vec, we use the same hyperparameters ($D = 128$, $R = 10$, $L = 80$, $\omega = 10$) and grid search over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ as mentioned in [19]. For the HONE variants, we set $D = 128$ and select the number of steps K automatically via a grid search over $K \in \{1, 2, 3, 4\}$ using 10% of the labeled data. We use all graphlets with 2-4 nodes and set $D_\ell = 16$ for the local motif embeddings unless otherwise mentioned. All methods use logistic regression (LR) with an L2 penalty. The model is selected using 10-fold cross-validation on 10% of the labeled data. Experiments are repeated for 10 random seed initializations. All data is from NetworkRepository [39]. Data statistics are provided in Table 1.

4.2 Link Prediction

Given a partially observed graph G with a fraction of missing edges, the link prediction task is to predict these missing edges. We generate a labeled dataset of edges. Following [19, 43], we obtain positive examples by removing 50% of edges randomly, whereas *negative examples* are generated by randomly sampling an equal number of node pairs that are not connected with an edge $(i, j) \notin E$. For each method, we learn embeddings using the remaining graph that consists of only positive examples. Using the embeddings from each method, we then learn a model to predict whether a given edge in the test set exists in E or not. To construct edge features from the node embeddings, we use the mean operator defined as $(\mathbf{z}_i + \mathbf{z}_j)/2$.

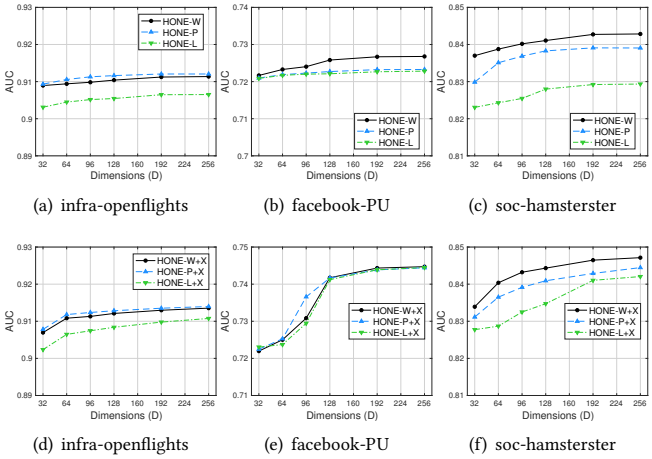
The AUC results are provided in Table 2. In all cases, the HONE methods outperform the other embedding methods with an overall mean gain of 19.19% (and up to 75.21% gain). In all cases, the gain achieved by the proposed HONE variants is significant at $p < 0.01$. We also derive a total ranking of the embedding methods over all graph problems based on average gain. Results are provided in the last column of Table 2. Among the five HONE variants in Table 2, we find that HONE-**W** and HONE-**P** perform the best overall.

Table 2: Prediction Results (AUC). See text for discussion.

	soc-hamster	*rt-twitter-cop*	*soc-wiki-Vote*	*tech-routers-rf*	*facebook-PU*	*inf-openflights*	*soc-bitcoinA*	R_{ANK}
HONE-W (Eq. 2)	0.841	0.843	0.811	0.862	0.726	0.910	0.979	**1**
HONE-P (Eq. 5)	0.840	0.840	0.812	0.863	0.724	0.913	0.980	**2**
HONE-L (Eq. 7)	0.829	0.841	0.808	0.858	0.722	0.906	0.975	**3**
HONE-$\widehat{\text{L}}$ (Eq. 8)	0.829	0.836	0.803	0.862	0.722	0.908	0.976	**5**
HONE-$\widehat{\text{L}}_{\text{rw}}$ (Eq. 9)	0.831	0.834	0.808	0.863	0.723	0.909	0.976	**4**
Node2Vec [19]	0.810	0.635	0.721	0.804	0.701	0.844	0.894	**6**
DeepWalk [34]	0.796	0.621	0.710	0.796	0.696	0.837	0.863	**7**
LINE [47]	0.752	0.706	0.734	0.800	0.630	0.837	0.780	**8**
GraRep [9]	0.805	0.672	0.743	0.829	0.702	0.898	0.559	**9**
Spectral [48]	0.561	0.699	0.593	0.602	0.516	0.606	0.629	**10**

4.3 Diffusion Variants

Now we investigate HONE variants that use diffusion (Section 2.6). These methods perform attribute diffusion using the k-step motif matrices (Section 2.6) and concatenate the resulting diffused features. Unless otherwise mentioned, we use linear diffusion defined in Eq. 19 with the default hyperparameters (Section 4.1). Note the initial matrix $\mathbf{X}$ described in Section 2.6 represents node motif counts derived by applying relational aggregates (sum, mean, and max) over each nodes local neighborhood and then scaled using Euclidean norm. We compare the HONE methods *with attribute diffusion* to the HONE methods without diffusion for link prediction. Results are reported in Table 3. The relative gain between each pair of HONE methods is computed for each graph and Table 3 reports the mean gain for each pair of HONE methods. Overall, we observe that HONE with attribute diffusion improves predictive performance in general. We also investigated other attribute diffusion variants from Section 2.6 and noticed similar results.



(a) infra-openflights (b) facebook-PU (c) soc-hamsterster

(d) infra-openflights (e) facebook-PU (f) soc-hamsterster

Figure 3: Experiments varying the dimensionality D of the node embeddings. See text for discussion.

Table 3: Mean gain of the HONE methods *with attribute diffusion* relative to each of the original HONE methods.

	HONE-W	HONE-P	HONE-L	HONE-$\widehat{\text{L}}$	HONE-$\widehat{\text{L}}_{\text{rw}}$
HONE-W + $\bar{\text{X}}$	0.76%	1.30%	1.38%	1.24%	1.08%
HONE-P + $\bar{\text{X}}$	1.58%	2.12%	2.20%	2.06%	1.90%
HONE-L + $\bar{\text{X}}$	0.62%	1.15%	1.23%	1.09%	0.93%
HONE-$\widehat{\text{L}}$ + $\bar{\text{X}}$	1.37%	1.91%	1.99%	1.85%	1.69%
HONE-$\widehat{\text{L}}_{\text{rw}}$ + $\bar{\text{X}}$	1.27%	1.81%	1.88%	1.74%	1.58%

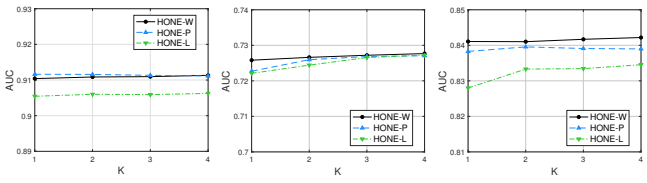
4.4 Visitor Stitching from Web Logs

We also evaluate HONE for the visitor stitching problem [25]. Given large-scale web log data, the visitor stitching task is to infer the web sessions that correspond to the same user/individual and stitch them. The resulting stitched data is often used in many important downstream applications and therefore accurately associating web sessions to the appropriate user helps reduce noise and avoid issues with sparisty while allowing for better personalization and recommendations [25]. We use two real-world visitor stitching data sets (web logs) from large companies (with known ground-truth). For each, we construct a network where nodes represent web sessions (of users) that are linked to IPs used in that session, web pages visited, and user agent information (device, O/S, browser). Company-A network has 8.9M nodes and 55.2M edges between them whereas Company-B has 22.8M nodes and 61.3M edges (Table 1). This data has known ground-truth node pairs. These are included as positive node pair examples. We also randomly sample the same number of negative examples. Afterwards, embeddings for each node pair are computed by taking the mean: $(\mathbf{z}_i + \mathbf{z}_j)/2$. We then learn a logistic regression (LR) model from the embeddings derived from the node pairs in the training set, and use it to predict the web sessions (nodes) that correspond to the same user in the held-out test set. We evaluate the methods using precision, recall, F1 score, and AUC. A good visitor stitching method should have very high precision (*e.g.*, 0.95-0.99) to avoid inappropriately stitching sessions from different users. However, it should also have good recall since a method that does not make many predictions is not very useful, even if they are accurate. We compare HONE to the previous baseline methods and use the same experimental setup described in Section 4.1. Experiments were performed on a MacBook Pro laptop with a 3.1 GHz Intel Core i7 processor and 16GB of memory. We report results for HONE-P and HONE-P+$\bar{X}$ (with diffusion). Similar results were observed using other HONE variants. In Table 4, we observe that HONE outperforms all other methods across the evaluation metrics. Notably, both HONE variants have very high precision (an important requirement for visitor stitching) with good recall. Hence, both methods accurately stitch a good amount of users while making only a very small number of mistakes (high precision). In comparison, the other baseline methods make significantly fewer predictions (lower recall) that are mostly of much lower quality. Both Node2Vec (N2V) and DeepWalk (DW) did not terminate within the 6 hour time limit. The results in Table 4 demonstrate the effectiveness of HONE for visitor stitching.

Table 4: Visitor stitching results for two real company data sets. ETL = Exceeded Time Limit (6 hours).

		N2V	DW	LINE	GraRep	Spec.	HONE	HONE+$\bar{\text{X}}$
Comp.-A	**Prec.**	ETL	ETL	0.8947	0.9924	0.7404	**0.9999**	0.9810
	Rec.	ETL	ETL	0.6254	0.4388	0.4907	0.6676	**0.8777**
	F1	ETL	ETL	0.7362	0.6085	0.5902	0.8007	**0.9265**
	AUC	ETL	ETL	0.7968	0.7215	0.5730	0.8572	**0.9304**
Comp.-B	**Prec.**	ETL	ETL	0.8362	0.9945	0.7731	**0.9923**	0.9885
	Rec.	ETL	ETL	0.3985	0.0937	0.2768	0.7336	**0.7736**
	F1	ETL	ETL	0.5397	0.1713	0.4077	0.8249	**0.8534**
	AUC	ETL	ETL	0.6843	0.5447	0.5259	0.8626	**0.8811**

*N2V=Node2Vec, DW=DeepWalk



(a) infra-openflights (b) facebook-PU (c) soc-hamsterster

Figure 4: Experiments comparing the predictive performance as the number of steps K changes.

4.5 Parameter Sensitivity

Effect of the dimensionality D of the embedding: To understand the effect of the number of embedding dimensions D on predictive performance, we set K=2 and vary the number of dimensions D. Results are reported in Figure 3. In general, predictive performance increases as a function of the embedding dimension D as observed in Figure 3. For small values of D, the predictive performance tends to increase more rapidly and then slows as D becomes large. For large values of D, the predictive performance for most graphs stabilizes reaching a near-plateau. As the embedding dimension D becomes large, the higher-order node embeddings get richer and the model parameter space expands becoming capable of capturing weaker signals/structural properties of nodes, at the risk of overfitting.

Effect of the number of steps K: To understand the effect of the number of steps K on predictive performance, we set D=128 and vary the number of steps $K \in \{1, 2, 3, 4\}$ used in the higher-order node embeddings. Results are shown in Figure 4. In general, the appropriate number of steps K appears to depend on the specific graph/structural properties and method used (Figure 4). For instance, the choice of K for HONE-L appears to be more important compared to the other HONE variants. Overall, we observe that HONE is relatively robust to changes in K. Due to space constraints, many results and plots had to be removed.

4.6 Runtime & Scalability

To evaluate the runtime performance and scalability of the proposed framework, we learn node embeddings for Erdös-Rényi graphs of increasing size (from 100 to 10 million nodes) such that each graph has an average degree of 10. In Figure 5, we observe that HONE is fast and scales linearly as the number of nodes increases. In addition, we also compare the runtime performance of HONE against node2vec [19] and LINE [47]. For the HONE variant, we use HONE-P with K=2. Default parameters are used for each method.

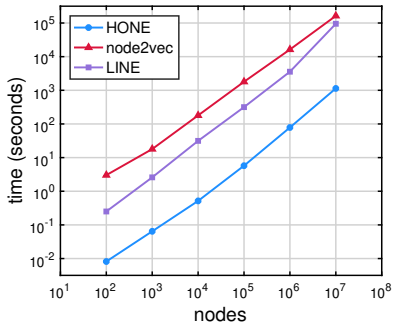


Figure 5: Runtime comparison. See text for discussion.

In Figure 5, HONE is shown to be significantly faster and more scalable than node2vec and LINE as the number of nodes increases. In particular, node2vec takes 1.8 days (45.3 hours) for 10 million nodes, while HONE finishes in only 19 minutes as shown in Figure 5. Strikingly, this is 143 times faster than node2vec. Alternatively, LINE takes 26.35 hours and thus HONE is 83x faster than LINE.

4.7 Visualization

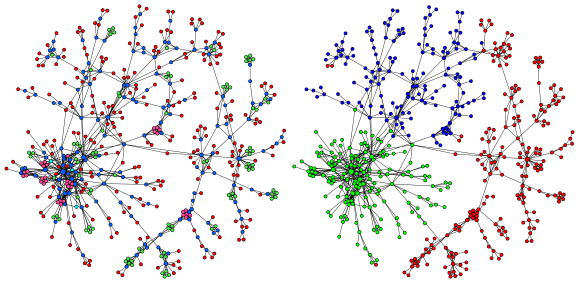
In these experiments, we validate the ability of HONE to learn embeddings that capture roles (based on structural characteristics) [40] as opposed to communities (based on proximity/closeness, density) [17]. Given a node embedding matrix from some method, we use k-means to assign nodes to clusters, repeat this 1000 times and take the minimum distance clustering identified. The number of clusters is selected using MDL. We then visualize the graph structure and color nodes by their cluster assignments.

To validate the ability of HONE to embed nodes with similar structural characteristics/behavior, we first use the well-studied Borgatti-Everett graph [8]. This graph is notable since there is a known exact role assignment and two obvious communities [40]. Furthermore, all nodes in this graph have the same degree. In Figure 6(a), HONE partitions nodes into three structurally equivalent classes (roles) whereas in Figure 6(b) node2vec partitions nodes into two communities based on proximity. Notably, the role assignment given by HONE is the known exact role assignment that has been extensively studied in the literature [8].

(a) HONE (b) node2vec

Figure 6: Validation of HONE's ability to capture roles on graphs with known ground-truth. See text for discussion.

We also investigate another network (diseasome) in Figure 7. Notably, HONE captures roles (groups of nodes with similar structural characteristics) [40] as shown in Figure 7(a) whereas node2vec reveals three clear communities as shown in Figure 7(b). Notably, the roles given by HONE in Figure 7(a) are intuitive and make sense. For instance, nodes assigned to the red role are peripheral nodes at the edge of the network whereas nodes assigned to the blue role act as hubs or gate-keepers connecting different groups of nodes.



(a) HONE (b) node2vec

Figure 7: Application of HONE for role discovery [40] in the diseasome network. See text for discussion.

5 RELATED WORK

Higher-order network motifs: This paper introduces the problem of *higher-order network (motif-based) embedding* and proposes a general framework for learning such embeddings based on higher-order connectivity patterns called graphlets. There has been one recent approach that used network motifs as base features for network representation learning [43]. However, that approach is different from the proposed framework as it focuses on learning inductive relational functions that represent compositions of relational operators applied to a base feature. Other methods use high-order network properties (such as graphlet frequencies) as features for graph classification [49], community detection [5, 6], and visualization and exploratory analysis [1]. However, this work focuses on network representation learning using network motifs. Recently, HOPE [33] was proposed based on the Katz index to measure higher-order node proximity. However, HOPE is fundamentally different since it is proximity-based and does not leverage motifs.

Node embeddings: There has been a lot of interest recently in learning node (and edge [2]) embeddings from large-scale networks automatically [3, 10, 19, 34, 37, 42, 43, 47]. Many node embedding methods [10, 19, 34, 37, 47] have leveraged the popular skip-gram model [13, 28]. These methods all use random walks to gather a sequence of node ids which are then used to learn node embeddings [10, 19, 34, 37, 47]. Other methods are based on "implicit walks" such as GraRep [9] (a generalization of LINE [47]) that incorporates node neighborhood information beyond 2-hops. However, these methods are all based on "higher-order" proximity/distance (and thus learn community-based embeddings as shown in [41]) as opposed to being based on higher-order network motifs for learning structural role-based embeddings. Graph Convolutional Networks (GCNs) adapt CNNs to graphs using Laplacian and spectral convolutions with a form of aggregation over the neighbors [15, 21, 26, 31]. These node embedding methods may also benefit from ideas developed in this work including the weighted motif Laplacian matrices described in Section 2.3. Other work has focused on incremental methods for spectral clustering [12].

Heterogeneous networks [46] have also been recently considered [11, 16] as well as attributed networks with labels [22, 23]. Huang *et al.* [23] proposed an approach for attributed networks with labels whereas Yang *et al.* [52] used text features to learn node representations. Liang *et al.* [27] proposed a semi-supervised approach for networks with outliers. Bojchevski *et al.* [7] proposed an unsupervised rank-based approach. There has also been some recent

work on semi-supervised network embeddings [26, 53] and methods for improving the learned representations [45, 50, 51]. A few work have begun to explore the problem of learning node embeddings from temporal networks [30, 36, 44]. This work is different from the problem discussed in this paper. More recently, an approach called role2vec was proposed that learns role-based node embeddings by first mapping each node to a type via a function and then uses the proposed notion of attributed (feature-based/labeled/typed) random walks to derive role-based embeddings for the nodes that capture structural similarity [3]. This approach was shown to generalize many existing random walk-based methods.

6 CONCLUSION

This work proposed *Higher-Order Network Embeddings* (HONE), a new class of structural node embedding methods that use higher-order network motifs to learn *structural role-based embeddings*. We described a general computational framework for learning higher-order (role-based) network embeddings that is flexible with many interchangeable components. The experiments demonstrated the effectiveness of HONE for a number of important tasks including link prediction and visitor stitching. Future work will investigate the framework using other useful motif-based matrix formulations.

REFERENCES

[1] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. 2015. Efficient Graphlet Counting for Large Networks. In *ICDM*. 10.

[2] Nesreen K. Ahmed, Ryan A. Rossi, Theodore L. Willke, and Rong Zhou. 2017. Edge Role Discovery via Higher-Order Structures. In *PAKDD*. 291–303.

[3] Nesreen K. Ahmed, Ryan A. Rossi, Rong Zhou, John Boaz Lee, Xiangnan Kong, Theodore L. Willke, and Hoda Eldardiry. 2018. Learning Role-based Graph Embeddings. In *arXiv:1802.02896*.

[4] Nesreen K. Ahmed, Theodore L. Willke, and Ryan A. Rossi. 2016. Estimation of Local Subgraph Counts. In *BigData*. 586–595.

[5] Alex Arenas, Alberto Fernandez, Santo Fortunato, and Sergio Gomez. 2008. Motif-based communities in complex networks. *J Phys A Math Theor.* 41, 22 (2008).

[6] Austin R Benson, David F Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166.

[7] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep Gaussian Embedding of Attributed Graphs: Unsupervised Inductive Learning via Ranking. *arXiv:1707.03815* (2017).

[8] S.P. Borgatti and M.G. Everett. 1992. Notions of position in social network analysis. *Sociological methodology* 22, 1 (1992), 1–35.

[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning graph representations with global structural information. In *CIKM*. ACM, 891–900.

[10] Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning community embedding with community detection and node embedding on graphs. In *CIKM*. 377–386.

[11] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *SIGKDD*. 119–128.

[12] Pin-Yu Chen, Baichuan Zhang, Mohammad Al Hasan, and Alfred O Hero. 2015. Incremental method for spectral clustering of increasing orders. In *arXiv:1512.07349*.

[13] Winnie Cheng, Chris Greaves, and Martin Warren. 2006. From n-gram to skipgram to concgram. *Int. J. of Corp. Linguistics* 11, 4 (2006), 411–433.

[14] Michael Collins, Sanjoy Dasgupta, and Robert E Schapire. 2002. A generalization of principal components analysis to the exponential family. In *NIPS*. 617–624.

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.

[16] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *SIGKDD*.

[17] S. Fortunato. 2010. Community detection in graphs. *Phy. Rep.* 486, 3-5 (2010).

[18] Gene H Golub and Charles F Van Loan. 2012. *Matrix computations*. JHU Press.

[19] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.

[20] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.

[21] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv:1506.05163* (2015).

[22] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *SDM*.

[23] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM*.

[24] Jingu Kim, Yunlong He, and Haesun Park. 2014. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. *Journal of Global Optimization* 58, 2 (2014), 285–319.

[25] Sungchul Kim, Nikhil Kini, Jay Pujara, Eunyee Koh, and Lise Getoor. 2017. Probabilistic visitor stitching on cross-device web logs. In *WWW*. 1581–1589.

[26] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[27] Jiongqian Liang, Peter Jacobs, and Srinivasan Parthasarathy. 2017. SEANO: Semi-supervised Embedding in Attributed Networks with Outliers. In *arXiv:1703.08100*.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR Workshop*. 10.

[29] Cameron Musco and Christopher Musco. 2015. Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *Advances in Neural Information Processing Systems*. 1396–1404.

[30] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-Time Dynamic Network Embeddings. In *WWW BigNet*.

[31] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *arXiv:1605.05273*.

[32] Jinoh Oh, Wook-Shin Han, Hwanjo Yu, and Xiaoqian Jiang. 2015. Fast and robust parallel SGD matrix factorization. In *SIGKDD*. ACM, 865–874.

[33] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *SIGKDD*. 1105–1114.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. 701–710.

[35] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinfo.* 23, 2 (2007), e177–e183.

[36] Mahmudur Rahman, Tanay Kumar Saha, Mohammad Al Hasan, Kevin S Xu, and Chandan K Reddy. 2018. DyLink2Vec: Effective Feature Representation for Link Prediction in Dynamic Networks. *arXiv:1804.05755* (2018).

[37] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. 2017. Struc2Vec: Learning Node Representations from Structural Identity. In *SIGKDD*.

[38] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. 2009. A randomized algorithm for principal component analysis. *SIAM J. Matrix Anal. Appl.* 31, 3 (2009).

[39] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. 4292–4293. http://networkrepository.com

[40] Ryan A. Rossi and Nesreen K. Ahmed. 2015. Role Discovery in Networks. *Transactions on Knowledge and Data Engineering* 27, 4 (April 2015), 1112–1131.

[41] Ryan A. Rossi, Di Jin, Sungchul Kim, Nesreen K. Ahmed, Danai Koutra, and John Boaz Lee. 2019. From Community to Role-based Graph Embeddings. In *arXiv:1908.08572*.

[42] Ryan A. Rossi, Luke K. McDowell, David W. Aha, and Jennifer Neville. 2012. Transforming graph data for statistical relational learning. *Journal of Artificial Intelligence Research* 45, 1 (2012), 363–441.

[43] Ryan A. Rossi, Rong Zhou, and Nesreen K. Ahmed. 2018. Deep Inductive Graph Representation Learning. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*. 14.

[44] Tanay Kumar Saha, Thomas Williams, Mohammad Al Hasan, Shafiq Joty, and Nicholas K Varberg. 2018. Models for Capturing Temporal Smoothness in Evolving Networks for Learning Latent Representation of Nodes. In *arXiv:1804.05816*.

[45] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *TNNLS* 20, 1 (2009), 61–80.

[46] Chuan Shi, Xiangnan Kong, Yue Huang, S Yu Philip, and Bin Wu. 2014. HeteSim: A General Framework for Relevance Measure in Heterogeneous Networks. *TKDE* 26, 10 (2014), 2479–2492.

[47] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*. 1067–1077.

[48] Lei Tang and Huan Liu. 2011. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery* 23, 3 (2011), 447–478.

[49] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. Graph kernels. *JMLR* 11 (2010), 1201–1242.

[50] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*. 1225–1234.

[51] Jason Weston, Frédéric Ratle, and Ronan Collobert. 2008. Deep learning via semi-supervised embedding. In *ICML*. 1168–1175.

[52] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network Representation Learning with Rich Text Information.. In *IJCAI*.

[53] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv:1603.08861* (2016).

[54] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, SVN Vishwanathan, and Inderjit Dhillon. 2014. NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion. *VLDB* 7, 11 (2014), 975–986.