Is a Single Embedding Enough? Learning Node Representations that Capture Multiple Social Contexts

Alessandro Epasto
Google AI
New York, NY
aepasto@google.com

Bryan Perozzi
Google AI
New York, NY
bperozzi@acm.org

ABSTRACT

Recent interest in graph embedding methods has focused on learning a single representation for each node in the graph. But can nodes really be best described by a single vector representation? In this work, we propose a method for learning multiple representations of the nodes in a graph (e.g., the users of a social network). Based on a principled decomposition of the ego-network, each representation encodes the role of the node in a different local community in which the nodes participate. These representations allow for improved reconstruction of the nuanced relationships that occur in the graph – a phenomenon that we illustrate through state-of-the-art results on link prediction tasks on a variety of graphs, reducing the error by up to 90%. In addition, we show that these embeddings allow for effective visual analysis of the learned community structure.

CCS CONCEPTS

• **Information systems** → **Data mining**; **Social networks**; • **Computing methodologies** → **Dimensionality reduction and manifold learning**; **Neural networks**; **Learning latent representations**; *Cluster analysis*;

KEYWORDS

graph embeddings; representation learning; polysemous representations

ACM Reference Format:
Alessandro Epasto and Bryan Perozzi. 2019. Is a Single Embedding Enough? Learning Node Representations that Capture Multiple Social Contexts. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313660

1 INTRODUCTION

Learning embedded representations of graphs is a recent and very active area [2, 14, 25, 37, 38, 44]. In a nutshell, an embedding algorithm learns a latent vector representation that maps each vertex v in the graph G to a *single* d dimensional vector. This area has found strong applications, as the embedding representation of nodes leads to improved results in data mining and machine learning tasks, such as node classification [37], user profiling [39], ranking [26], and link prediction [2, 25]. In virtually all cases, the crucial assumption of the embedding methods developed so far is that a *single* embedding vector has to be learned for each node in the graph. Thus, the embedding method can be said to seek to identify the single role or position of each node in the geometry of the graph.

This observation allows us to draw a historical parallel between the very recent research area of graph embedding and the more established field of community detection or graph clustering [22, 24]. Detecting clusters[1] in real world networks is a central topic in computer science, which has an extensive literature. At the beginning of its development, graph clustering has focused mostly on the study of non-overlapping clustering methods [24, 43]. In such methods, each node is assigned to a *single* cluster or community. While the problem of non-overlapping clustering is better understood and has found strong applications and theoretical results, recently, much attention has been devoted to developing *overlapping* clustering methods [16, 20, 40], where each node is allowed to participate in multiple communities. This interest in overlapping comunities is motivated by a number of recent observations of real world networks [1, 16, 19, 29, 30] that show a lack of clear (non-overlapping) community structure.

These findings motivate the following research question: can embedding methods benefit from the awareness of the overlapping clustering structure of real graphs? In particular, can we develop methods where nodes are embedded in multiple vectors, representing their participation in different communities?

In this paper, we provide positive results for these two questions. We develop Splitter, an unsupervised embedding method that allows nodes in a graph to have multiple embeddings to better encode their participation in multiple overlapping communities. Our method is based on recent developments in ego-net analysis, in particular, in overlapping clustering algorithms based on ego-network partitioning [20]. More precisely, we exploit the observation in [16, 19, 20, 40] that cluster structure is easier to identify at the local level. Intuitively, this happens because each node interacts with a given neighbor in usually a single context (even if it is part of many different commmunities in total).

Splitter extends this idea to the case of node embeddings. In particular, we exploit the *persona graph* concept defined by Epasto et al. [20]. This method, given a graph G, creates a new graph G_P (called the persona graph of G), where each node u in G is represented by a series of replicas. These replica nodes, (called the *persona*(s) of u) in G_P, represents an instantiation of the node u in the local community to which it belongs. The method was originally introduced to obtain overlapping clusters. In this paper, we instead show that ego-net based techniques can lead to improvements in

[1]Note that in the paper, we use the terms "cluster" and "community" interchangeably.

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313660

embedding methods as well. In particular, we demonstrate that a natural embedding method based on the persona graph outperforms many embedding baselines in the task of link prediction.

To summarize, the contributions of this paper are as follows:

(1) We introduce Splitter, a novel graph embedding method that embeds each node in the graph into multiple embedding vectors, which is based on the analysis of the overlapping structure of communities in real networks.
(2) Our method adds a novel graph regularization constraint to the optimization that enforces consistency of the multiple learned representations for each node.
(3) The method automatically determines the number of embeddings used for each node depending on a principled analysis of the local neighborhood of the node. This does not require the user to specify the number of embeddings as a parameter.
(4) We show experimentally strong improvements over several embedding baselines for the important task of link prediction.
(5) We show how our method enables visual discovery of community membership for nodes that are part of multiple social groups.

2 METHOD

In this section we describe Splitter, our proposed method for learning multiple community-aware representations for a node. First, we start with a review of the preliminaries necessary to understand the work in Section 2.1. Next, in Section 2.2 we discuss our extension for learning multiple node representations. Then we introduce Splitter in Section 2.3, and close with discussing some details of the optimization in Section 2.4.

2.1 Preliminaries

Our method builds upon recent work related to node decomposition [20], and learning node representations with neural networks [35, 37, 38]. Here we describe the basics of both methods.

2.1.1 Notation. We begin with some notation. Let $G = (V, E)$ be an undirected graph[2] consisting of a set V of nodes and a set $E \subset V \times V$ of edges. Let $G[U] = (U, E \cap U \times U)$ be the induced graph of a subset of G's nodes, $U \subset V$. Given a node $u \in V$, we denote its neighborhood as the set of nodes connected to it $N_u = \{v; (u, v) \in E\}$, and its *ego-network* (or ego-net) as the graph induced on the neighborhood $G[N_u]$. We note that the ego-net of u does not include the node u itself in this definition. Finally, let $\mathcal{A}$ be a *non-overlapping clustering* algorithm that given G as an input, returns a partition $\mathcal{A}(G) = (V_1, \ldots, V_t)$ of the vertices V into t disjoint sets (let $\mathrm{np}_{\mathcal{A}}(G) = t$ denote the number of partitions in output.).

2.1.2 Persona Decomposition. The topic of community detection and graph clustering has been of great interest to the community over the last several decades. While much work has focused on finding large clusters, it has been noted that while the community detection problem is hard at scale (the *macroscopic* level), it is relatively trivial when viewed locally (the *microscopic* level) [17, 19]. Using this intuition, a recent proposal from Epasto et al. [20] uses the clusters found in the ego-network of a node (its neighbors and their induced subgraph) as the basis to define a new graph, the *persona graph*, G_P. The nodes in this new graph, which are called *personas*, divide the interactions of each original node in G into several semantic subgroups, which capture different components (or senses) of its network behavior.

More formally, let us assume that we are given a graph G and a clustering algorithm $\mathcal{A}$. The persona decomposition (as proposed in [20]) employs the following algorithm PersonaGraph$(G, \mathcal{A})$ to transform G to its persona graph G_P:

(1) For each node $u \in V$, we use the clustering algorithm $\mathcal{A}$ to partition the ego-net of u. Let $\mathcal{A}(G[N_u]) = \{N_u^1, N_u^2, \ldots, N_u^{t_u}\}$, where $t_u = \mathrm{np}_{\mathcal{A}}(G[N_u])$.
(2) Create a set V' of personas. Each node v_o in V will correspond to t_{v_o} personas (the number of splits of the ego-net of v_o) in V_P, denoted by v_i for $i = 1, \ldots, t_{v_o}$.
(3) Add edges between personas. If $(u, v) \in E$, $v \in N_u^i$ and $u \in N_v^j$, then add an edge (u_i, v_j) to E_P.

After using this procedure, one obtains the persona graph G_P which has some interesting properties. First, every node in G_P is a node from the original graph, split into one or more personas. However, there is no additional connectivity information – the number of edges in G_P is equal to the number of edges in the persona graph. This means that the space required to store G_P is (almost) the same as the original graph. Second, each node in the original graph can be mapped to its corresponding persona(s). However, the community structure of G_P can be wildly different from the original graph. Standard clustering methods, when run on G_P instead genereate overlapping clusterings of G. This phenomena of exposing differing clustering information is visualized further in Section 4.

2.1.3 Graph Embedding. Before introducing our method for learning multiple embeddings for each node, we first review the standard setting of network representation learning, in which a single embedding is learned for each node. The purpose of network embedding is to learn a mapping $\Phi\colon v \in V \mapsto \mathbb{R}^{|V| \times d}$, which encodes the latent structural role of a node in the graph. This, in practice, can be achieved by representing the mapping Φ as a $|V| \times d$ matrix of free parameters that are learned by solving an optimization problem. Perozzi et al. [37] first introduced a modeling of the vertex representation that encodes the node as a function of its co-occurrences with other nodes in short truncated random walks.

More precisely, the method consists of performing multiple random walks over the social graph from each node. The sequences of these walks are then used to extract the co-occurrences of nodes in short sub-windows. These co-occurrences capture the diffusion in the neighborhood around each vertex in the graph, and explore the local community structure around a node. More concretely, the goal of the embedding method is to learn a representation that enables an estimate of the likelihood of a vertex v_i co-occurring with other nodes in the sub-window of a short random walk:

$$\Pr\left(v_i \mid \left(\Phi(v_1), \Phi(v_2), \cdots, \Phi(v_{i-1})\right)\right) \tag{1}$$

[2]The method can be also defined for directed graphs in the obvious way; however, we describe it for undirected graphs for simplicity

Notice that the exact computation of this conditional probability is computationally expensive for increasing lengths of the random walks, so DeepWalk uses two techniques to address this challenge. First, the order of the neighboring vertices is ignored. Second, the method reverses the learning task; instead of predicting a missing vertex using the context, it addresses the opposite problem of predicting its local structure using the vertex itself.

These modifications result in the following optimization problem for computing the vertex representations of each node in DeepWalk:

$$\underset{\Phi}{\text{minimize}} \quad -\log \Pr\left(\{v_{i-w}, \cdots, v_{i+w}\} \setminus v_i \mid \Phi(v_i)\right) \tag{2}$$

In the DeepWalk [37] model, the probability of a node v_i co-occurring with v_j is estimated by using a softmax to map the pairwise similarity to a probability space,

$$Pr(v_i|v_j) = \frac{exp(\Phi(v_i) \cdot \Phi'(v_j))}{\sum_{j \in \mathcal{V}} exp(\Phi(v_i) \cdot \Phi'(v_j)} \tag{3}$$

Where $\Phi'(v_i)$ and $\Phi'(v_j)$ represent the "input" and "output" embeddings for node v_i and v_j respectively [34].

2.2 Learning Multiple Node Representations

As discussed so far, network representation learning seeks to learn a function that maps each node to its own representation. Here, we discuss our modifications that were made in light of the fact that we wish to learn one or more representation for each node.

Using the persona decomposition discussed in Section 2.1.2, we can convert the input graph G into the persona graph G_P. From here, it seems like a straightforward application of existing methods to learn one representation for each node $v \in |V_P|$, and as such, learn one or more representation for each original node $u \in |V|$. Unfortunately, this is not the case. The strength of the persona decomposition is also a weakness - it can create a graph that is quite different from that of the original input. In fact, the persona graph can be so different that it may consist of many disconnected components, even if the original graph was connected! Thus, these disconnected components can cause difficulties for representation learning methods. To address these challenges, we propose two improvements to the learning algorithm.

First, we propose adding a constraint, that the persona representations be able to predict their original node in addition to predicting the personas around it in G_P. Specifically, given a persona v_i, we propose to require its representation include a dependency on the node v_o in the original graph G:

$$\Pr\left(v_o \mid \Phi_{G_P}(v_i)\right). \tag{4}$$

To control the strength of our graph regularization, we introduce the parameter λ, which combines with Eq. (2) to yield the following optimization problem:

$$\begin{aligned}\underset{\Phi_{G_P}}{\text{minimize}} \quad & -\log \Pr\left(\{v_{i-w}, \cdots, v_{i+w}\} \setminus v_i \mid \Phi_{G_P}(v_i)\right) \\ & -\lambda \log \Pr\left(v_o \mid \Phi_{G_P}(v_i)\right).\end{aligned} \tag{5}$$

Put another way, this change to the optimization enforces that there are invisible edges to each persona's parent, informing the learning process and regularizing the degree to which the persona representations can deviate. This effectively lets the model reason about the different connected components that may exist after the persona transformation. We note that in practice, we achieved good results on all graphs we considered by simply setting $\lambda = 0.1$.

Secondly, we propose to also make the representation $\Phi_{G_P}(v)$ of a node v's personas depend on its original representation $\Phi_G(v)$ as a prior via initialization. Initializing all personas to the same position in $\mathbb{R}^d$, combined with the regularization term from Eq. (5), constrains the persona embeddings to behave like cohesive parts of a single entity. We note that this does not mean that all of a node's personas end up in the same position at the end of the optimization! Instead, we find that personas with similar roles stay close together, while personas with different roles separate. This is discussed further in Section 4.2, where we examine a visualization of Splitter embeddings. Finally, there is an additional benefit of using the representation of the original graph Φ_G as an initialization – it can help avoid potentially bad random initializations, which can lower task performance [14].

Inference with Multiple Representations. Notice that our method outputs multiple embeddings for each node. To do inference of node features or to predict edges between nodes, one can use standard ML methods to learn a function of the multiple embeddings of each node (or pair of nodes). However, basic aggregations can work too. In our experiments with link prediction, we simply use the maximum dot product over all pairs of embeddings of u and v to predict the likelihood of the pair of nodes being connected.

2.3 Splitter

Using the ideas discussed so far, we present the details of our approach.

2.3.1 Parameters. In addition to an undirected graph $G(V, E)$, the clustering algorithm $\mathcal{A}$ used to obtain the persona graph, as well as the dimensionality of the representations d, our algorithm uses a number of parameters that control the embedding learning process. The first group of parameters deal with sampling G, an essential part of any graph embedding process. Without loss of generality, we describe the parameters to control the sampling process in the notation of Perozzi et. al [37]. Briefly, they are w, the window size to slide over the random walk, t, the length of the random walk to sample from each vertex, and γ the number of walks per vertex. We emphasize that our approach is not limited to simple uniform random walk sampling - any of the more recently proposed graph sampling strategies [25, 35, 38] can be applied in the Splitter model.

The next group of parameters control the optimization. The parameter α controls the learning rate for stochastic gradient descent, and λ effects how strongly the original graph representation regularizes the persona learning.

Finally, an embedding function EmbedFn is used to learn a representation Φ_G of the nodes in the original graph. In order to use the learning algorithm that we specify, this embedding method simply needs to produce representations where the dot-product between vectors encodes the similarity between nodes. The most popular graph embedding methods meet this criteria, including DeepWalk [37], LINE [44], and node2vec [25].

Algorithm 1 SPLITTER. Our method for learning multiple representations of nodes in a graph

Input:
$G(V, E)$, a graph
w, window size
d, embedding size
γ, walks per vertex
t, walk length
α, learning rate
λ, graph regularization coefficient
$\mathcal{A}$, clustering algorithm for the ego-nets
EMBEDFN, a graph embedding method which uses the dot-product similarity (e.g. DeepWalk, node2vec)

Output:
Φ_{G_P} a matrix with one or more representations for each node
$P2N$, a mapping of the rows of Φ_{G_P} to V (the original nodes)

1: **function** SPLITTER(G, EMBEDFN)
2: $G_P \leftarrow$ PERSONAGRAPH$(G, \mathcal{A})$ ▷ Create the persona graph G_P of G using clustering algorithm $\mathcal{A}$ and method in [20].
3: $P2N \leftarrow \emptyset$
4: $\Phi_G \leftarrow$ EMBEDFN(G, w, d, γ, t) ▷ Embed original graph
5: **for each** $v_o \in V$ **do**
6: **for each** $v_j \in$ personas of v_o **do**
7: $\Phi_{G_P}(v_j) \leftarrow \Phi_G(v_o)$ ▷ Initialize j-th persona of v_o
8: $P2N(v_j) \leftarrow v_o$
9: **for** $i = 0$ to γ **do**
10: $O = \text{Shuffle}(V_{G_P})$
11: **for each** $v_i \in O$ **do**
12: $\mathcal{W}_{v_i} = RandomWalk(G_P, v_i, t)$
13: **for each** $v_j \in \mathcal{W}_{v_i}$ **do**
14: **for each** $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
15: $J_{G_P}(\Phi_{G_P})_= - \log \Pr(u_k \mid \Phi(v_j))$
16: $J_G(\Phi_{G_P})_= - \log \Pr(P2N(v_j) \mid \Phi(v_j))$
17: $\Phi_{G_P} = \Phi_{G_P} - \alpha * \left(\frac{\partial J_{G_P}}{\partial \Phi_{G_P}} + \lambda \frac{\partial J_G}{\partial \Phi_{G_P}} \right)$
18: **return** Φ_{G_P}, $P2N$

2.3.2 Algorithm. Here, we describe our full algorithm, shown in Algorithm 1 in detail. Lines 2-4 initialize the data structures, create the persona graph, and learn the embedding of the underlying graph. We note that not all of the nodes will necessarily be split; this will depend on each ego-net's structure, as described in Section 2.1.2. Lines 5-8 use the persona graph to initialize the persona representations Φ_{G_P}. The remainder of the algorithm (lines 9-17) details how the persona representations are learned. Line 12 generates the random walks to sample the context of each vertex. WLOG can be graph samples generated in any meaningful way - including uniform random walks [37], skipped random walks [38], or random walks with backtrack [25]. Line 15 calculates the loss due to nodes in the persona graph (how well the persona representation of Φ_{v_j} is able to predict the observed node u_k). Line 16 computes the loss due to the graph regularization (how well the persona representation Φ_{v_j} is able to predict its corresponding original node). Finally, line 18 returns the induced representations Φ_{G_P} and their mapping back to the original nodes $P2N$.

Complexity. The complexity of the algorithm is dominated by two major steps: creating the persona graph and the Skip-gram model learning. Both parts have been analyzed in previous works [14, 20], so we report briefly on the complexity of our algorithm here. Suppose the clustering algorithm $\mathcal{A}$ used on the ego-nets has a complexity of $T(m')$ for analyzing an ego-net of m' edges. Further, suppose the original graph has $\mathcal{T}$ triangles. As such, the persona creation method has a total running time $O(\mathcal{T} + m + \sqrt{m}T(m))$. Moreover, as a worst case, $\mathcal{T} = O(m^{3/2})$ the complexity is $O(m^{3/2} + \sqrt{m}T(m))$. Suppose, for instance, that a linear time clustering algorithm is used; then, the total cost of this phase as a worst case is $O(m^{3/2})$. However, as observed before [20], the algorithm scales much better in practice than this pessimistic bound because the number of triangles is usually much smaller than $m^{3/2}$. The embedding method for a graph of n' total nodes in the persona graph has instead a complexity $O(n'\gamma t w d \log(n')))$, as shown in [14], where d is the number of dimensions, t is the walk length size, w is the window size and γ is the number of random walks used. Notice, that $n' \in O(m)$, as each node u can have at most $O(deg(u))$ persona nodes, so the worst case complexity is: $O(m^{3/2} + \sqrt{m}T(m) + m\gamma t w(d + d\log(m)))$.

2.4 Optimization

As detailed in [37], using representations to predict the probability of nodes (Line 15-16, Algorithm 1) is computationally expensive. To deal with this, we use the hierarchical softmax [34] to calculate these probabilities more efficiently. For completeness's sake, we remark that an alternative optimization strategy exists (using noise contrastive estimation [34]).

Thus, our model parameter set includes both Φ_{G_P} and T, the parameters used internally by the tree in the hierarchical softmax. Further, we use the back-propagation algorithm to estimate the derivatives in lines 15-16, and a stochastic gradient descent (SGD) to optimize the model's parameters. The initial learning rate α for SGD is set to 0.025 at the beginning of the training and then decreased linearly with the number of nodes we have seen so far. Additionally, the parameter λ regularizes the persona embeddings by how much they deviate from the node's original representation.

3 TASK: LINK PREDICTION

In this section, we study how the SPLITTER model proposed so far can be used for the task of link prediction – judging the strength of a potential relationship between the two nodes. We focus on this fundamental application task for the following reasons.

First, we are interested in developing new models that can capture the variation of social behaviors that are expressed in real-world social networks (such as membership in multiple communities). Several recent works suggest that link prediction (or *network reconstruction*) [2, 50] is the best way to analyze an unsupervised network embedding's performance, as it is a primary task (unlike node classification - a secondary task that involves a labeling process that may be uncorrelated with the graph itself). Second, there are several important industrial applications of link prediction on

real networks (e.g. friend suggestion on a social network, product recommendation on an e-commerce site, et cera).

Finally, this task highlights a particular strength of our method. Splitter's ability to model the differing components of a node's social profile (its *personas*) make it especially suitable for the task of link prediction. This addresses a fundamental weakness of most node embedding methods, which effectively treat a node's representation as an average of its multiple senses – a representation that may not make sense in continuous space. Unlike previous work utilizing community information in embeddings [12, 47, 51], we aim to expose the nuanced relationships in a network by sub-dividing the nodes (not the macro-scale community relationships found by joining nodes together).

3.1 Experimental Design

3.1.1 Datasets. We test our Splitter method as well as other baselines on a dataset of five directed and undirected graphs. Our datasets are all publicly available: PPI is introduced [25, 42], while the other datasets are from Stanford SNAP library [28]. For each dataset, in accordance with the standard methodology used in the literature [2, 25], we use the largest weakly connected component of the original graph. We now provide statistics for our dataset.

Directed graphs:

(1) **soc-epinions**: A social network $|V| = 75,877$ and $|E| = 508,836$. Edges represent who trusts whom in the opinion-trust dataset of Epinions.
(2) **wiki-vote**: A voting network $|V| = 7,066$ and $|E| = 103,663$. Nodes are Wikipedia editors. Each directed edges represents a vote for allowing another user becoming an administrator.

Undirected graphs:

(1) **ca-HepTh**: Arxiv's co-author network of High Energy Physics Theory $|V| = 9,877$ and $|E| = 25,998$. Each edge represents co-authorship between two author nodes.
(2) **ca-AstroPh**: Arxiv's co-author network of Astrophysics, $|V| = 17,903$ and $|E| = 197,031$. Each edge represents co-authorship between two author nodes.
(3) **PPI**: Protein-protein interaction graph, $|V| = 3,852$ and $|E| = 20,881$. This represents a natural dataset, where each node is a protein and there is an edge betwen two proteins if they interact (more details in [25]).

3.2 Task

The Link Prediction task follows the methodology introduced in [2, 25], which we briefly detail here. First, the input graph is split into two edge sets, E_{train} and E_{test}, of equal size. The test edges are removed uniformly at random, with the restriction that they do not disconnect the graph. E_{test} is then used as positive examples for a classification task. A corresponding equal sized set of non-existent (random) edges are generated to use as negative examples for testing. The baseline methods are providing the training edges as input, which they use to learn a similarity model (embedded or otherwise). The performance of each method is then measured by ranking the removed edges. Specifically, for each method, we report the ROC-AUC.

3.3 Methods

Here we describe the numerous baseline methods we tested against, including both traditional non-embedding baselines (such as common neighbors) and several embedding baselines. We also detail the application of Splitter's multiple representations of this task.

Non-embedding Baselines: Here, we report standard methods for link prediction that are solely based on the analysis of the adjacency matrix of the graph and in particular, on the immediate neighborhood of the nodes in the graph. These methods take into input E_{train} during inference. Thus, we denote $N(u)$ as the neighbors of u observed in E_{train}. For directed graphs, $N(u)$ only refers to the outgoing edges. In the non-embedding baseline considered, we score an edge (u, v) as a $g(u, v)$, which is a function of $N(u)$ and $N(v)$ only. We consider the following baselines:

(1) **Jaccard Coefficient (J.C.) :**

$$g(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

(2) **Common Neighbors (C.N.):**

$$g(u, v) = |N(u) \cap N(v)|$$

(3) **Adamic Adar (A.A.):**

$$g(u, v) = \sum_{x \in N(u) \cap N(v)} \frac{1}{\log(|N(x)|)}$$

We apply these methods to both the original graph and the persona graph. For the persona graph, we follow a technique similar to our Splitter method to extract a single score from the pairwise similarity (say Jaccard Coefficient) of the multiple persona nodes of a pair of nodes u,v. We also define the Jaccard Coefficient of u, v in the persona graph as the maximum Jaccard Coefficient of a persona node of u and a persona node of v in the persona graph. Similarly, we define the baselines Common Neighbors and Adamic Adar in the persona graph. We report results for using the maximum as aggregation function consistently with our Splitter method, but we experimented as well with many other functions, such as the minimum and the mean and we observed the maximum to perform best.

Embedding Baselines: We also consider the following embedding baselines. These methods take as input E_{train} to learn embedding $\Phi_G(u)$ for every graph node u. During inference, then only the learned embedding are used but *not* the original graph. We compare against these state-of-the-art embedding methods:

(1) Laplacian **EigenMaps** [6] determines the lowest eigenvectors of the graph Laplacian matrix.
(2) **node2vec** [25] learns the embedding by performing random walks on the training edges E_{train} and learning a minimizing skipgram objective (Equation 3).
(3) **DNGR** [11] performs a non-linear (i.e. deep) node embeddings passing a "smoothed" adjancency matrix through a deep auto-encoder. The "smoothing" (called Random Surfing) an alternative to random walks, which effectively has a different context weight from node2vec.
(4) **Asymmetric** [2] is a recent method that learns embeddings by explicitly modeling the edges that appear in the graph.

	Dataset	Non-Embedding Adjacency Methods						Embedding Methods						
		Original graph			Persona graph				Embedding Baselines					Ours
		J.C.	C.N.	A.A.	J.C.	C.N.	A.A.	d	Eigen Maps	node2vec	DNGR	Asymmetric	M-NMF	Splitter
directed	soc-epinions	0.649	0.649	0.647	0.797	0.797	0.797	8	†	0.725	†	0.695	†	**0.972**
								16	†	0.726	†	0.699	†	**0.974**
								32	†	0.714	†	0.700	†	**0.973**
								64	†	0.699	†	0.698	†	**0.970**
								128	†	0.691	†	0.718	†	**0.967**
	wiki-vote	0.579	0.580	0.562	0.860	0.865	0.866	8	0.613	0.643	0.630	0.608	0.886	**0.950**
								16	0.607	0.642	0.622	0.643	0.912	**0.952**
								32	0.600	0.641	0.619	0.683	0.926	**0.953**
								64	0.613	0.642	0.598	0.702	0.932	**0.952**
								128	0.622	0.643	0.554	0.730	0.934	**0.939**
undirected	ca-HepTh	0.765	0.765	0.765	0.553	0.553	0.553	8	0.786	0.731	0.706	0.605	0.852	**0.877**
								16	0.790	0.787	0.780	0.885	0.884	**0.897**
								32	0.795	0.858	0.829	0.884	0.903	**0.909**
								64	0.802	0.886	0.868	0.870	0.912	**0.917**
								128	0.812	0.901	0.897	0.820	0.908	**0.920**
	ca-AstroPh	0.942	0.942	0.944	0.874	0.874	0.874	8	0.825	0.811	0.852	0.592	0.903	**0.959**
								16	0.825	0.833	0.877	0.657	0.935	**0.972**
								32	0.825	0.899	0.917	0.942	0.954	**0.978**
								64	0.824	0.934	0.939	0.936	0.966	**0.982**
								128	0.829	0.955	0.968	0.939	0.974	**0.985**
	PPI	0.766	0.776	0.779	0.698	0.701	0.702	8	0.710	0.733	0.583	0.550	0.739	**0.865**
								16	0.711	0.707	0.687	0.786	0.776	**0.869**
								32	0.709	0.691	0.741	0.794	0.793	**0.869**
								64	0.707	0.671	0.767	0.813	0.817	**0.866**
								128	0.737	0.698	0.769	0.799	0.840	**0.863**

Table 1: We report the ROC-AUC for a link prediction task performed using an ablation test. The columns J.C., C.N. and A.A. stand for the baselines jaccard coefficient, common neighbors and adamic-adar, respectively.
The rows represents the datasets (directed and undirected) while the columns represent the methods compared. We compare our Splitter method with three non-embeddings methods (applied to both the original and persona graph) as well four embeddings baselines. For the embedding methods we report results using dimensionality $\{8, 16, 32, 64, 128\}$. We report in bold the highest AUC-ROC for each dimension and dataset. Notice that for Splitter the dimension refers to the size of the embedding of each persona node (i.e. the total embedding size of each node is larger). The next table reports results at parity of space. Results with † indicate lack of completion. We used a machine with 32 GB ram.

Dataset	Avg. Personas per Node $\bar{p}$
soc-epinions	3.03
wiki-vote	4.00
ca-HepTh	2.39
ca-AstroPh	2.53
ppi	4.97
19159	

Table 2: Average number of persona nodes per node in original graph.

We compare against the most similar model proposed in the work, the shallow asymmetric model.

(5) **M-NMF** [47] uses a modularity based community detection model to jointly optimize the embedding and community assignment of each node. Unlike Splitter, this method assigns each node to one community, and is based on joining nodes together (not splitting them apart).

We run each of these methods with their suggested default parameters.[3]

During inference with the baselines, we use the embedding of a pair of node u and v to rank the likelihood of the link u, v formed by employing a scoring function that takes in the input the embeddings of the two nodes. To do so, for consistency with previous work, we used the same methodology of [2], which we summarize

[3]In order to advance the field, and ensure the reproducibility of our method, we are releasing an implementation of Splitter at https://github.com/google-research/google-research/tree/master/graph_embedding/persona.

here. Let Y_u and Y_v be, respectively, the embeddings of u and v. The edge scoring function is defined as follows: for **EigenMaps**, it is $-||Y_u - Y_v||$; for **node2vec**, we use the off-shelve binary classification LogisticRegression algorithm of sklearn to lean a model over the Hadamard product of the embeddings of the two nodes; for **DNGR**, we use the bottleneck layer values as the embeddings and the dot product as similarity; for **Asymmetric**, we use the dot product; and for **M-NMF**, similarly to node2vec, we train a model on the Hadamard product of the embeddings.

Our Method (Splitter): In order to use Splitter for link prediction, we need a method to calculate a single similarity score between two nodes (u,v) in the original graph G, each of which may have multiple persona representations. Specifically, as the Splitter embedding model uses the dot-product to encode the similarity between the two node's representations, we need a method to extract a single score from the pairwise similarity of the (potentially) multiple persona nodes. Similar to applying non-embedding baselines to the persona graph, we experimented with a number of aggregation functions (including *min*, *max*, *mean*, etc). The highest performing aggregation function was the maximum, so we define the similarity between the two nodes in G to be the maximum dot-product between any of their constituent personas in G_P.

For learning embeddings with Splitter, we set the random walk length $t = 40$, number of walks per node $\gamma = 10$, and the window size $w = 5$, the initial learning rate $\alpha = 0.025$, and the graph regularization coefficient $\lambda = 0.1$. For EmbedFn, we used node2vec with random walk parameters ($p = q = 1$) which is equivalent to DeepWalk.

3.4 Experimental Results

In the following table, we report the experimental comparison of Splitter with several embedding and non-embedding baselines. For each experiment, we report the AUC-ROC in a link prediction task performed using an ablation test described in Section 3.2. For consistency of comparison, we use the experimental settings (datasets and training/testing splits) of [2] for the baselines. Hence, the baselinesâĂŹ numbers are the same of [2] and are reported for completeness.

We first report in Table 1 the results of Splitter with several dimensionality settings. All results involving Splitter or the adjacency matrix on persona graph baseline uses the connected component method for ego-net clustering. We chose the connected component algorithm for easy replication of the results because it performs very well, as well as due to its previous use in ego-net clustering [20, 40]. In particular [20], showed theoretical results in a graph model for the connected component method at ego-net level.

We first take a look at the adjacency matrix baselines. Here, we consider both the vanilla version of the well-known baselines in the original graph, as well as the application of such baselines on the persona pre-processed graph (with a methodology similar to Splitter, as described above). We observe that simply applying the persona preprocessing to the standard baselines does not consistently improve the results over using them in the original graph. In particular, we only observe improvements in two of the five graphs we analyzed, while sometimes, we see even strong losses in applying this simple pre-processing, especially for our sparsest graphs, such as ca-HepTh. This confirms that the gains observed in our Splitter do not come merely from the pre-possessing.

Now, we consider the embedding methods. In this table, to gain an understanding of Splitter embeddings, we compare different sizes of Splitter embeddings (per persona node) with same size embeddings of other methods (per node). Before delving into the results, a note is important; since each node can have multiple persona embeddings, the total embedding size of Splitter (for the same dimension) can be larger than that of another standard embedding method. For this reason, we will later compare the results at same total embedding size. First, we observe in Table 1 that at the same level of dimensionality, Splitter always outperforms all other baselines. The improvement is particularly significant in the largest graph *epinions* where our method using size 8 embeddings improves AUC-ROC by a factor of 40% (reduction in error of 90%) even when compared with the best baseline with 128 dimensions. Similarly, our method achieves close to optimal performances in two of the other largest graphs, wiki-vote and ca-AstroPh.

As we have mentioned before, our method embeds each node into multiple embeddings (one for each persona node), so for a given dimension d of the embedding, the average embedding size per node is given by $d\bar{p}$, where $\bar{p}$ is the average number of personas per nodes (i.e. the average number of ego-net clusters per node) in the graph. We report the average number of persona nodes for all the graphs in our datasets in Table 2. As we notice, the average number of persona nodes is between 2 and 5 in our datasets (using the connected component ego-network splitting algorithm). We report in Table 3 a different look at the previous results. This time, we compare Splitter's AUC-ROC to other embeddings, allowing for the same (or higher) total embedding space. In our example in Table 3, we fix $d = 16$ for the Splitter method and then we compute the effective average embedding size for each dataset ($\bar{p}16$). Thereafter, we compare our results with the best result for each baseline that uses approximately $16\bar{p}$ dimensions (for fairness of comparison, we actually round $16\bar{p}$ to the next power of 2 and always allow more space for the other methods vs our method). Thus, it is possible to observe that the AUC-ROC of Splitter in is higher than that of every other baseline using about $16\bar{p}$ dimensions for all datasets, except once (in ca-HepTh, M-NMF is better, and we observe this is the sparsest graph). This confirms that the method improves over the baselines even after accounting for the increased space due to the presence of multiple embeddings for nodes. We observe the same results for other d besides $d = 16$.

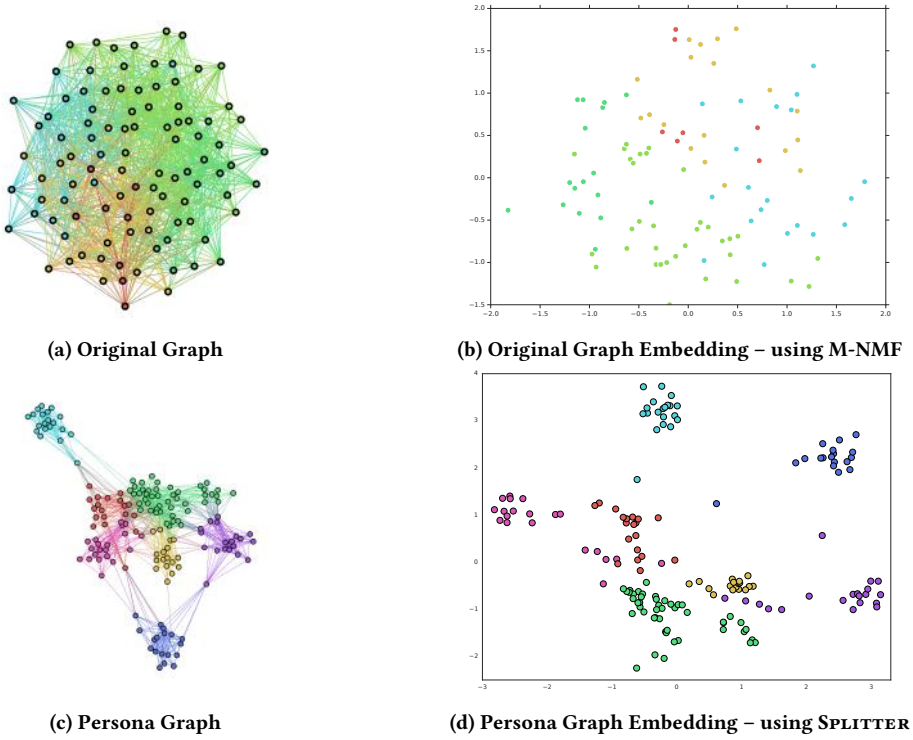
4 TASK: VISUALIZATION

4.1 Synthetic graphs

To gain insight on how our embedding method framework operates, we first provide a visualization of a small synthetic graph. The methodology we use is similar to that of [20], which we report for completeness. We created a random graph with planted overlapping communities using the Lanchinetti et al [27] model. We chose this model for consistency with previous ego-net based works [16, 20] and because the model replicates several properties of real-world graphs, such as power law distribution of degrees,

Dataset	$d_{max} = 16\bar{p}$	Best EigenMaps	Best Node2Vec	Best DNGR	Best Asymmetric	Best M-NMF	Splitter $d = 16$
soc-epinions	48.5	†	0.726	†	0.700	†	**0.974**
wiki-vote	64.0	0.613	0.643	0.630	0.702	0.932	**0.952**
ca-HepTh	38.2	0.802	0.886	0.868	0.885	**0.912**	0.897
ca-AstroPh	40.5	0.824	0.934	0.939	0.942	0.966	**0.972**
ppi	79.5	0.737	0.733	0.769	0.813	0.840	**0.869**

Table 3: AUC-ROC of Splitter with $d = 16$ compared with best baseline allowed larger total embedding space (at approximate space parity).



(a) Original Graph

(b) Original Graph Embedding – using M-NMF

(c) Persona Graph

(d) Persona Graph Embedding – using Splitter

Figure 1: Original graph and Persona graph with corresponding embeddings. Notice how the persona graph community structure is clearer than the one in the original graph and this corresponds to more separated embeddings. The colors in the original and persona graph corresponds to community found by a modularity based algorithm. Left side pictures are used with permission from Epasto et al. [20]. (best viewed in color)

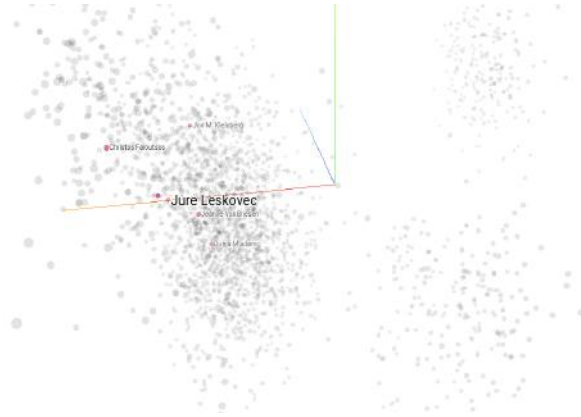
varying community sizes and the membership of nodes in varying community numbers. The graph contains 100 nodes and has 9 highly overlapping ground-truth communities.

We show the results of the visualization in Figure 1. First in Figure 1a and Figure 1c we show a force-directed layout of the original graph and the corresponding persona graph (using Gephi [5] with the same visualization settings). The node coloring corresponds to the discovered communities using a non-overlapping community detection that optimizes modularity [7]. As observed by Epasto et al. [20] on this dataset, the persona graph has a much clearer community structure, finding 8 of the 9 overlapping communities (only 5 communties around found in the original graph).

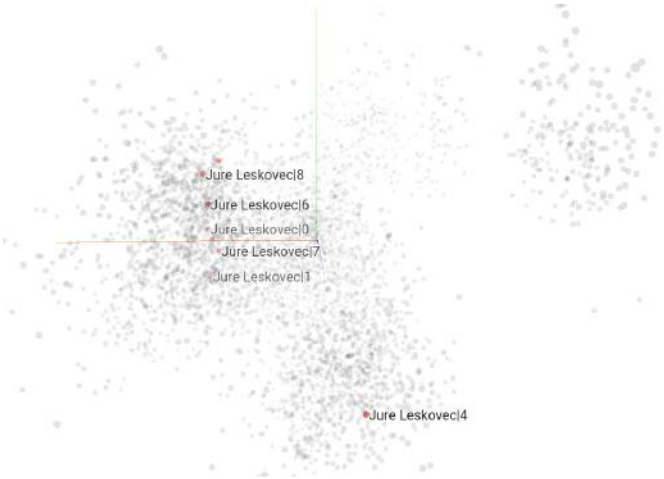
We now turn our attention to the embeddings output by our method. In Figure 1b, we show a 2D embedding obtained using M-NMF [47] with default settings on the original graph, while in Figure 1d, we show a 2D embedding obtained by our method. As such, it is possible to appreciate how the Splitter embeddings more clearly identify the community structure, as the eight communities found are better separated. In contrast, the M-NMF embeddings do not show a clear separation for this graph, which has highly overlapping communities.

4.2 DBLP Co-Authorship Graph

We then turn our attention to a real-world co-authorship graph using DBLP data. The `4area` graph contains co-authorship relationships extracted from papers in 4 areas of study: Data Mining, Machine Learning, Databases, and Information Retrieval [36]. It is possible to see in Figure 2a and Figure 2b, respectively, a plot of



(a) Node2vec – Highlighted Author



(b) Splitter Embeddings – Highlighted Author

Figure 2: Comparison of embedding visualization of a node2vec and Splitter in a DBLP co-authorship graph containing authors from 4 main areas: Data Mining, Machine Learning, Data Base and Information Retrieval. It is possible to observe how the 4 areas correspond to (more or less) well separated regions in the space in both embedding methods. However, standard embeddings force each author that participates in multiple communities to be represented in a single point, while the persona methods allow author-nodes to be represented by multiple embeddings. We show one such embedding for a prolific author with contributions in both the Data Mining and Machine Learning community. Notice how the node2vec embedding results in the author being embedding in the data mining region only (this is best viewed in color).

node2vec embeddings and of Splitter embeddings. Notice how in Figure 2b there are 4 areas more or less separated that upon inspection, corresponds to the 4 different fields. A similar observation is possible for Figure 2a, representing node2vec embeddings. The key observation in this application scenario is that many authors (in particular the most prolific ones) can contribute to more than one area of study or more than one subarea. However, standard embedding methods force each node to be embedded in only one point in the space, while the Splitter method allows to represent a node as a combination of embeddings (in this setting, we obtain 1.58 personas per node on average). In the Figures 2a and 2b, respectively, we highlight the embeddings learned for one such prolific author, `Jure Leskovec`. Note how this author has one single embedding obtained by node2vec, and multiple embeddings given in output by Splitter. Upon inspection, we observe that the author is embedded in a data mining region by node2vec, surrounded by other prominent authors in Data Mining, such as `Christos Faloutsos`.

However, when observing the representations learned through our Splitter method, we see that this author has a number of persona representations. Moreover, many of the personas reflect different sub-groups of coauthors in Data Mining that our node (e.g. one persona corresponds to co-authorship with other students while at CMU). These personas encode significant portions of the 'average' or 'global' position, which is captured by node2vec. Nevertheless, we also see that a persona ((Jure Leskovec|4)) is now present in the Machine Learning cluster. This illustrates how the representations from Splitter allows the node to span both the

Data Mining and Machine Learning region of the space, better characterizing the contributions of the node. Similar observations hold for other authors.

5 RELATED WORK

Our work bridges two very active areas of research: ego-net analysis and graph embeddings. As these are vast and fast growing, we will restrict ourselves to reviewing only the most closely related papers in these two areas.

5.1 Graph embedding

These methods learn one embedding per graph node, with an objective that maximizes (minimizes) the product (distance) of node embeddings if they are 'close' in the input graph. These are most related to our work. In fact, our work builds on the approach introduced by DeepWalk [37], which learns node embeddings using simulated random walks. This idea has been extended to consider node embeddings learned on different variations of random walks [25], as well as other graph similarity measures [45], other loss functions [8], or additional information such as edge labels [15]. These node embeddings have been used as features for various tasks on networks, such as node classification [37], user profiling [39], and link prediction [2, 25, 50]. More recent work in the area has examined preserving the structural roles of nodes in a network [41, 46], learning embeddings for a graph kernel [4], or proposing attention methods to automatically learn the model's hyperparameters [3]. For more information on node embedding, we direct the reader to a recent survey [13].

Moreover, most node embedding methods focus on only learning one representation for each node in the graph. Walklets [38] decompose the hierarchy of relationships exposed in a random walk into a family of related embeddings. However, this is distinctly different from our work, as each node is represented exactly once at each level of the hierarchy. In addition, the representations are learned independently from each other. HARP [14] is a meta-approach for finding good initializations for embedding algorithms. As a by-product, it produces a series of representations that encode a hierarchical clustering. A number of other works focus on learning community representations, or using communities to inform the node embedding process [12, 47, 51]. Unlike these works, which focus on aggregating nodes into *less* representations, we focus on *dividing* nodes into more representations. This allows our approach, Splitter, to more easily represent prolific nodes that may have overlapping community membership.

5.2 Ego-net analysis

Our work is most closely related to the line of research in social network analysis based on ego-net clustering. From their introduction by Freeman [23] in 1982, ego-nets or ego-networks are a mainstay of social-network analysis [9, 18, 21, 48]. Rees and Gallagher [40] jump-started a rich stream of works [16, 19, 20] that exploit ego-network level clusters to extract important structural information on communities [22], to which a node belongs. They proposed to partition nodes' ego-net-minus-ego graphs in their connected components to find a global overlapping clustering of the graph. Coscia et al. [16] improved over their clustering method by proposing to use a more sophisticated label propagation ego-net partitioning technique. Several authors have since built on such a line of work to improve the scalability and accuracy of ego-net based clustering [10, 19, 20, 33, 33], while others have designed ego-net analysis methods that tackle user metadata on top of the ego-net connectivity [31, 32, 49].

Mostly related to our work is the recent paper by Epasto el al. [20], where they introduce the persona graph method for overlapping clustering. They present a scalable overlapping clustering algorithm based on a local ego-net partition. Their algorithm first creates the ego-nets of all nodes and partition them (in parallel) using any non-overlapping algorithm. These ego-net level partitions are then used to create the persona graph, which is described in more detail in this paper, as this is the basis of our embedding methods. Then, the persona graph is partitioned with another parallel clustering algorithm to obtain overlapping clusters in the original graph.

6 CONCLUSIONS

We introduced Splitter, a novel graph embedding method that builds on recent advances in ego-network analysis and overlapping clustering. In particular, we exploited the recently introduced persona graph decomposition to develop an embedding algorithm that represents nodes in the graph with multiple vectors in a principled way. Our experimental analysis shows strong improvements for the tasks of link prediction and visual discovery and exploration of the community membership of nodes.

Our method draws a connection between the rich and well-studied field of overlapping community detection and the more recent one of graph embedding which we believe may result in further research results. As future work we want to explore more in this direction, focusing on the following challenges: (1) exploiting embeddings for overlapping clustering; (2) studying the effect of this method on web-scale datasets; (3) developing theoretical results on this method; (4) applying our embeddings for classification and semi-supervised learning tasks; and (5) developi

REFERENCES

[1] Bruno Abrahao, Sucheta Soundarajan, John Hopcroft, and Robert Kleinberg. 2014. A separability framework for analyzing community structure. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8, 1 (2014), 5.

[2] Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. 2017. Learning Edge Representations via Low-Rank Asymmetric Projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17)*. ACM, New York, NY, USA, 1787–1796.

[3] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A. Alemi. 2018. Watch Your Step: Learning Node Embeddings via Graph Attention. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* 9198–9208. http://papers.nips.cc/paper/8131-watch-your-step-learning-node-embeddings-via-graph-attention

[4] Rami Al-Rfou, Dustin Zelle, and Bryan Perozzi. 2019. DDGK: Learning Graph Representations for Deep Divergence Graph Kernels. In *Proceedings of the 2019 World Wide Web Conference on World Wide Web, WWW 2019.*

[5] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. 2009. Gephi: an open source software for exploring and manipulating networks. (2009).

[6] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems.* 585–591.

[7] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[8] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *ICLR* (2018).

[9] R. Burt. 1995. *Structural Holes: The Social Structure of Competition.* Harvard Press.

[10] Nazar Buzun, Anton Korshunov, Valeriy Avanesov, Ilya Filonenko, Ilya Kozlov, Denis Turdakov, and Hangkyu Kim. 2014. EgoLP: Fast and Distributed Community Detection in Billion-Node Social Networks. In *2014 IEEE ICDM Workshops*. 533–540.

[11] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[12] 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 *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 377–386.

[13] Haochen Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2018. A tutorial on network embeddings. *arXiv preprint arXiv:1808.02590* (2018).

[14] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. Harp: Hierarchical representation learning for networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[15] Haochen Chen, Xiaofei Sun, Yingtao Tian, Bryan Perozzi, Muhao Chen, and Steven Skiena. 2018. Enhanced Network Embeddings via Exploiting Edge Labels. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*. 1579–1582.

[16] Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. 2014. Uncovering Hierarchical and Overlapping Communities with a Local-First Approach. *TKDD* (2014).

[17] Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. 2014. Uncovering hierarchical and overlapping communities with a local-first approach. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9, 1 (2014), 6.

[18] R. I. M. Dunbar and S. G. B. Roberts. 2010. Communication in Social Networks: Effects of Kinship, Network Size and Emotional Closeness. *Personal Relationships* (2010).

[19] Alessandro Epasto, Silvio Lattanzi, Vahab Mirrokni, Ismail Oner Sebe, Ahmed Taei, and Sunita Verma. 2015. Ego-net community mining applied to friend suggestion. *VLDB* 9, 4 (2015), 324–335.

[20] Alessandro Epasto, Silvio Lattanzi, and Renato Paes Leme. 2017. Ego-Splitting Framework: from Non-Overlapping to Overlapping Clusters. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 145–154.

[21] M. Everett and S. P. Borgatti. 2005. Ego network betweenness. *Social Networks* (2005).

[22] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* (2010).

[23] L. C. T. Freeman. 1982. Centered graphs and the structure of ego networks. *Mathematical Social Sciences* (1982).

[24] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.

[25] A. Grover and J. Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[26] Chin-Chi Hsu, Yi-An Lai, Wen-Hao Chen, Ming-Han Feng, and Shou-De Lin. 2017. Unsupervised Ranking using Graph Structures and Node Attributes. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 771–779.

[27] Andrea Lancichinetti and Santo Fortunato. 2009. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E* 80, 1 (2009), 016118.

[28] J. Leskovec and A. Krevl. 2014. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data

[29] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2009. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics* (2009).

[30] Jure Leskovec, Kevin J Lang, and Michael Mahoney. 2010. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*. ACM, 631–640.

[31] Jure Leskovec and Julian J Mcauley. 2012. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*. 539–547.

[32] Rui Li, Chi Wang, and Kevin Chen-Chuan Chang. 2014. User profiling in an ego network: co-profiling attributes and relationships. In *Proceedings of the 23rd international conference on World wide web*. ACM, 819–830.

[33] Panagiotis Liakos, Alexandros Ntoulas, and Alex Delis. 2016. Scalable link community detection: A local dispersion-aware approach. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 716–725.

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[35] Bryan Perozzi. 2016. *Local Modeling of Attributed Graphs: Algorithms and Applications*. Ph.D. Dissertation. State University of New York at Stony Brook.

[36] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. 2014. Focused clustering and outlier detection in large attributed graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1346–1355.

[37] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Knowledge Discovery and Data Mining*.

[38] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena. 2017. Don't Walk, Skip! Online Learning of Multi-scale Network Embeddings. In *2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*.

[39] Bryan Perozzi and Steven Skiena. 2015. Exact age prediction in social networks. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 91–92.

[40] Bradley S Rees and Keith B Gallagher. 2010. Overlapping community detection by collective friendship group inference. In *2010 International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 375–379.

[41] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 385–394.

[42] C. Stark, B.J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. 2006. BioGRID: A General Repository for Interaction Datasets. In *Nucleic Acids Research*. https://www.ncbi.nlm.nih.gov/pubmed/16381927

[43] Peter R Suaris and Gershon Kedem. 1988. An algorithm for quadrisection and its application to standard cell placement. *IEEE Transactions on Circuits and Systems* 35, 3 (1988), 294–303.

[44] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[45] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. (2018), 539–548.

[46] Ke Tu, Peng Cui, Xiao Wang, Philip S. Yu, and Wenwu Zhu. 2018. Deep Recursive Network Embedding with Regular Equivalence. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18)*. ACM, New York, NY, USA, 2357–2366.

[47] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community preserving network embedding. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[48] S. Wasserman and K. Faust. 1994. *Social network analysis: methods and applications*. Cambridge University Press.

[49] Jaewon Yang, Julian J. McAuley, and Jure Leskovec. 2014. Detecting cohesive and 2-mode communities indirected and undirected networks. In *WSDM*.

[50] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-Order Proximity Preserved Network Embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18)*. ACM, New York, NY, USA, 2778–2786.

[51] Vincent W Zheng, Sandro Cavallari, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2016. From node embedding to community embedding. *arXiv preprint arXiv:1610.09950* (2016).