GEMSEC: Graph Embedding with Self Clustering

Benedek Rozemberczki, Ryan Davies, Rik Sarkar and Charles Sutton
School of Informatics, The University of Edinburgh
{benedek.rozemberczki,ryan.davies}@ed.ac.uk,{rsarkar,csutton}@inf.ed.ac.uk

***Abstract*—Modern graph embedding procedures can efficiently process graphs with millions of nodes. In this paper, we propose *GEMSEC* – a graph embedding algorithm which learns a clustering of the nodes simultaneously with computing their embedding. *GEMSEC* is a general extension of earlier work in the domain of sequence-based graph embedding. *GEMSEC* places nodes in an abstract feature space where the vertex features minimize the negative log-likelihood of preserving sampled vertex neighborhoods, and it incorporates known social network properties through a machine learning regularization. We present two new social network datasets and show that by simultaneously considering the embedding and clustering problems with respect to social properties, *GEMSEC* extracts high-quality clusters competitive with or superior to other community detection algorithms. In experiments, the method is found to be computationally efficient and robust to the choice of hyperparameters.**

***Index Terms*—community detection, clustering, node embedding, network embedding, feature extraction.**

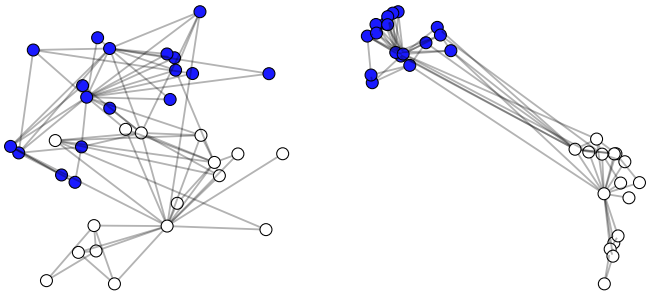
I. Introduction

Community detection is one of the most important problems in network analysis due to its wide applications ranging from the analysis of collaboration networks to image segmentation, the study of protein-protein interaction networks in biology, and many others [1], [2], [3]. Communities are usually defined as groups of nodes that are connected to each other more densely than to the rest of the network. Classical approaches to community detection depend on properties such as graph metrics, spectral properties and density of shortest paths [4]. Random walks and randomized label propagation [5], [6] have also been investigated.

Embedding the nodes in a low dimensional Euclidean space enables us to apply standard machine learning techniques. This space is sometimes called the *feature space* – implying that it represents abstract structural features of the network. Embeddings have been used for machine learning tasks such as labeling nodes, regression, link prediction, and graph visualization, see [7] for a survey. Graph embedding processes usually aim to preserve certain predefined differences between nodes encoded in their embedding distances. For social network embedding, a natural priority is to preserve community membership and enable community detection.

Recently, sequence-based methods have been developed as a way to convert complex, non-linear network structures into formats more compatible with vector spaces. These methods sample sequences of nodes from the graph using a randomized mechanism (e.g. random walks), with the idea that nodes that are "close" in the graph connectivity will also frequently appear close in a sampling of random walks. The methods then proceed to use this random-walk-proximity information as a basis to embed nodes such that socially close nodes are placed nearby. In this category, *Deepwalk* [8] and *Node2Vec* [9] are two popular methods.

While these methods preserve the proximity of nodes in the graph sense, they do not have an explicit preference for preserving social communities. Thus, in this paper, we develop a machine learning approach that considers clustering when embedding the network and includes a parameter to control the closeness of nodes in the same community. Figure 1(a) shows the embedding obtained by the standard *Deepwalk* method, where communities are coherent, but not clearly separated in the embedding. The method described in this paper, called *GEMSEC*, is able to produce clusters that are tightly embedded and separated from each other (Fig. 1(b)).



(a) DeepWalk (b) GEMSEC

Fig. 1. Zachary's Karate club graph [10]. White nodes: instructor's group; blue nodes: president's group. GEMSEC produces embedding with more tightly clustered communities.

A. Our Contributions

GEMSEC is an algorithm that considers the two problems of embedding and community detection simultaneously, and as a result, the two solutions of embedding and clustering can inform and improve each other. Through iterations, the embedding converges toward one where nodes are placed close to their neighbors in the network, while at the same time clusters in the embedding space are well separated.

The algorithm is based on the paradigm of sequence-based node embedding procedures that create d dimensional feature representations of nodes in an abstract feature space. Sequence-based node embeddings embed pairs of nodes close to each other if they occur frequently within a small window of each other in a random walk. This problem can be formulated as minimizing the negative log-likelihood of observed neighborhood samples (Sec. III) and is called the skip-gram

optimization [11]. We extend this objective function to include a clustering cost. The formal description is presented in Subsection III-A. The resulting optimization problem is solved with a variant of mini-batch gradient descent [12].

The detailed algorithm is presented in Subsection III-B. By enforcing clustering on the embedding, *GEMSEC* reveals the natural community structure (e.g. Figure 1).Our approach improves over existing methods of simultaneous embedding and clustering [13], [14], [15] and shows that community sensitivity can be directly incorporated into the skip-gram style optimization to obtain greater accuracy and efficiency.

In social networks, nodes in the same community tend to have similar groups of friends, which is expressed as high neighborhood overlap. This fact can be leveraged to produce clusters that are better aligned with the underlying communities. We achieve this effect using a regularization procedure – a smoothness regularization added to the basic optimization achieves more coherent community detection. The effect can be seen in Figure 3, where a somewhat uncertain community affiliation suggested by the randomized sampling is sharpened by the smoothness regularization. This technique is described in Subsection III-C.

In experimental evaluation we demonstrate that *GEMSEC* outperforms – in clustering quality – the state of the art neighborhood based [8], [9], multi-scale [16], [17] and community aware embedding methods [13], [14], [15]. We present new social datasets from the streaming service Deezer and show that the clustering can improve music recommendations. The clustering performance of *GEMSEC* is found to be robust to hyperparameter changes, and the runtime complexity of our method is linear in the size of the graphs.

To summarize, the main contributions of our work are:

1) *GEMSEC*: a sequence sampling-based learning model which learns an embedding of the nodes at the same time as it learns a clustering of the nodes.
2) Clustering in *GEMSEC* can be aligned to network neighborhoods by a smoothness regularization added to the optimization. This enhances the algorithm's sensitivity to natural communities.
3) Two new large social network datasets are introduced – from Facebook and Deezer data.
4) Experimental results show that the embedding process runs linearly in the input size. It generally performs well in quality of embedding and in particular outperforms existing methods on cluster quality measured by modularity and subsequent recommendation tasks.

We start with reviewing related work in the area and relation to our approach in the next section. A high-performance Tensorflow reference implementation of *GEMSEC* and the datasets that we collected can be accessed online[1].

II. Related Work

There is a long line of research in metric embedding – for example, embedding discrete metrics into trees [18] and into vector spaces [19]. Optimization-based representation of networks has been used for routing and navigation in domains such as sensor networks and robotics [20], [21]. Representations in hyperbolic spaces have emerged as a technique to preserve richer network structures [22], [23], [24].

Recent advances in node embedding procedures have made it possible to learn vector features for large real-world graphs [8], [16], [9]. Features extracted with these *sequence-based node embedding* procedures can be used for predicting social network users' missing age [7], the category of scientific papers in citation networks [17] and the function of proteins in protein-protein interaction networks [9]. Besides supervised learning tasks on nodes the extracted features can be used for graph visualization [7], link prediction [9] and community detection [13].

Sequence based embedding commonly considers variations in the sampling strategy that is used to obtain vertex sequences – truncated random walks being the simplest strategy [8]. More involved methods include second-order random walks [9], skips in random walks [17] and diffusion graphs [25]. It is worth noting that these models implicitly approximate matrix factorizations for different matrices that are expensive to factorize explicitly [26].

Our work extends the literature of node embedding algorithms which are community aware. Earlier works in this category did not directly extend the skip-gram embedding framework. *M-NMF* [14] applies computationally expensive non-negative matrix factorization with a modularity constraint term. The procedure *DANMF* [15] uses hierarchical non-negative matrix factorization to create community-aware node embeddings. *ComE* [13] is a more scalable approach, but it assumes that in the embedding space the communities fit a gaussian structure, and aims to model them by a mixture of Gaussians. In comparison to these methods, *GEMSEC* provides greater control over community sensitivity of the embedding process, it is independent of the specific neighborhood sampling methods and is computationally efficient.

III. Graph Embedding with Self Clustering

For a graph $G = (V, E)$, a node embedding is a mapping $f : V \rightarrow \mathbb{R}^d$ where d is the dimensionality of the embedding space. For each node $v \in V$ we create a d dimensional representation. Alternatively, the embedding f is a $|V| \times d$ real-valued matrix. In sequence-based embedding, sequences of neighboring nodes are sampled from the graph. Within a sequence, a node v occurs in the *context* of a window ω within the sequence. Given a sample S of sequences, we refer to the collection of windows containing v as $N_S(v)$. Earlier works have proposed random walks, second-order random walks or branching processes to obtain $N_S(v)$. In our experiments, we used unweighted first and second-order random walks for node sampling [8], [9].

Our goal is to minimize the negative log-likelihood of observing neighborhoods of source nodes conditional on feature

[1] https://github.com/benedekrozemberczki/GEMSEC

vectors that describe the position of nodes in the embedding space. Formally, the optimization objective is:

$$\min_{f} \quad \sum_{v \in V} -\log P(N_S(v)|f(v)) \tag{1}$$

for a suitable probability function $P(\cdot|\cdot)$. To define this P, we consider two standard properties (see [9]) expected of the embedding f in relation to N_S. First, it should be possible to factorize $P(N_S(v)|f(v))$ in line with *conditional independence* with respect to $f(v)$. Formally:

$$P(N_S(v)|f(v)) = \prod_{n_i \in N_S(v)} P(n_i \in N_S(v) \mid f(v), f(n_i)). \tag{2}$$

Second, it should satisfy *symmetry in the feature space*, meaning that source and neighboring nodes have a symmetric effect on each other in the embedding space. A softmax function on the pairwise dot products of node representations with $f(v)$ to get $P(n_i \in N_S(v) \mid f(v), f(n_i))$ express such a property:

$$P(n_i \in N_S(v) \mid f(v), f(n_i)) = \frac{\exp(f(n_i) \cdot f(v))}{\sum_{u \in V} \exp(f(u) \cdot (f(v))}. \tag{3}$$

Substituting (2) and (3) into the optimization function, we get:

$$\min_{f} \sum_{v \in V} \left[\ln \left(\sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right]. \tag{4}$$

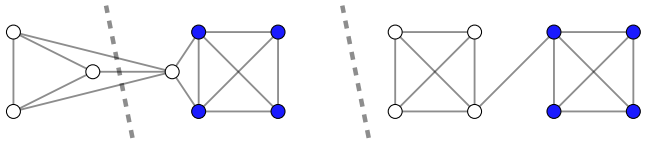
The partition function in Equation (4) enforces nodes to be embedded in a low volume space around the origin, while the second term forces nodes with similar sampled neighborhoods to be embedded close to each other.

A. Learning to Cluster

Next, we extend the optimization to pay attention to the clusters it forms. We include a clustering cost similar to k-means, measuring the distance from nodes to their cluster centers. This augmented optimization problem is described by minimizing a loss function over the embedding f and position of cluster centers μ, that is, $\min_{f,\mu} \mathcal{L}$, where:

$$\mathcal{L} = \underbrace{\sum_{v \in V} \left[\ln \left(\sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right]}_{\text{Embedding cost}} + \underbrace{\gamma \cdot \sum_{v \in V} \min_{c \in C} \|f(v) - \mu_c\|_2}_{\text{Clustering cost}}. \tag{5}$$

In Equation (5) we have C the set of cluster centers – the c^{th} cluster mean is denoted by μ_c. Each of these cluster centers is a d-dimensional vector in the embedding space. The idea is to minimize the distance from each node to its nearest cluster center. The weight coefficient of the clustering cost is given by the hyperparameter γ. Evaluating the partition function in the proposed objective function for all of the source nodes has a $\mathcal{O}(|V|^2)$ runtime complexity. Because of this, we approximate the partition function term with negative sampling which is a form of noise contrastive estimation [11], [27].



(a) Node capture. (b) Empty initialization.

Fig. 2. Potential issues with cluster cost weighting and cluster initialization. Different node colors denote different ground truth community memberships and the computed cluster boundary is denoted by the dashed line. In Subfigure 2a a single white node is captured in a cluster with the blue nodes due to clustering weight γ being high. In Subfigure 2b an empty cluster is initialized with no nodes in it. It is plausible that the cluster center remains empty throughout the optimization process.

$$\frac{\partial \mathcal{L}}{\partial f(v^*)} = \underbrace{\frac{\sum_{u \in V} \exp(f(v^*) \cdot f(u)) \cdot f(u)}{\sum_{u \in V} \exp(f(v^*) \cdot f(u))}}_{\text{Partition function gradient}} - \underbrace{\sum_{n_i \in N_S(v^*)} f(n_i)}_{\text{Neighbor direction}} + \underbrace{\gamma \cdot \frac{f(v^*) - \mu_c}{\|f(v^*) - \mu_c\|_2}}_{\text{Closest cluster direction}} \tag{6}$$

The gradients of the loss function in Equation 5 are important in solving the minimization problem. As a result we can obtain the gradients for node representations and cluster centers. Examining in more detail, the gradient of the objective function $\mathcal{L}$ with respect to the representation of node $v^* \in V$ is described by Equation (6) if μ_c is the closest cluster center to $f(v^*)$.

The gradient of the partition function pulls the representation of v^* towards the origin. The second term moves the representation of v^* closer to the representations of its neighbors in the embedding space while the third term moves the node closer to the closest cluster center. If we set a high γ value the third term dominates the gradient. This will cause the node to gravitate towards the closest cluster center which might not contain the neighbors of v^*. An example is shown in Figure 2a. If the set of nodes that belong to cluster center c is V_c, then the gradient of the objective function with respect to μ_c is described by

$$\frac{\partial \mathcal{L}}{\partial \mu_c} = -\gamma \cdot \sum_{v \in V_c} \frac{f(v) - \mu_c}{\|f(v) - \mu_c\|_2}. \tag{7}$$

In Equation 7 we see that the gradient moves the cluster center by the sum of coordinates of nodes in the embedding space that belong to cluster c. Second, if a cluster ends up empty it will not be updated as elements of the gradient would be zero. Because of this, cluster centers and embedding weights are initialized with the same uniform distribution. A wrong initialization just like the one with an empty cluster in Subfigure 2b can affect clustering performance considerably.

Data: $\mathcal{G} = (V, E)$ – Graph to be embedded.
N – Number of sequence samples per node.
l – Length of sequences.
ω – Context size.
d – Number of embedding dimensions.
$|C|$ – Number of clusters.
k – Number of noise samples.
γ_0 – Initial clustering weight coefficient.
α_0, α_F – Initial and final learning rate.
Result: $f(v)$, where $v \in V$
μ_c, where $c \in C$

1 Model $\leftarrow$ **Initialize Model**$(|V|, d, |C|)$
2 $t \leftarrow 0$
3 **for** n in *1:N* **do**
4 $\quad \widehat{V} \leftarrow$ **Shuffle**(V)
5 $\quad$ **for** v in $\widehat{V}$ **do**
6 $\quad\quad t \leftarrow t + 1$
7 $\quad\quad \gamma \leftarrow$ **Update** γ $(\gamma_0, t, w, l, N, |V|)$
8 $\quad\quad \alpha \leftarrow$ **Update** α $(\alpha_0, \alpha_F, t, w, l, N, |V|)$
9 $\quad\quad$ Sequence $\leftarrow$ **Sample Nodes**$(\mathcal{G}, v, l)$
10 $\quad\quad$ Features $\leftarrow$ **Extract Features**(Sequence, ω)
11 $\quad\quad$ **Update Weights**(Model, Features, γ, α, k)
12 $\quad$ **end**
13 **end**

Algorithm 1: GEMSEC training procedure

B. GEMSEC algorithm

We propose an efficient learning method to create *GEMSEC* embeddings which is described with pseudo-code by Algorithm 1. The main idea behind our procedure is the following. To avoid the clustering cost overpowering the graph information (as in Fig. 2a), we initialize the system with a low weight $\gamma_0 \in [0, 1]$ for clustering, and through iterations anneal it to 1.

The embedding computation proceeds as follows. The weights in the model are initialized based on the number of vertices, embedding dimensions and clusters. After this, the algorithm makes N sampling repetitions in order to generate vertex sequences from every source node. Before starting a sampling epoch, it shuffles the set of vertices. We set the clustering cost coefficient γ (line 7) according to an exponential annealing rule described by Equation (8). The learning rate is set to α (line 8) with a linear annealing rule (Equation (9)).

$$\gamma = \gamma_0 \cdot \left(10^{\frac{-t \cdot \log_{10} \gamma_0}{w \cdot l \cdot |V| \cdot N}}\right) \tag{8}$$

$$\alpha = \alpha_0 - (\alpha_0 - \alpha_F) \cdot \frac{t}{w \cdot l \cdot |V| \cdot N} \tag{9}$$

The sampling process reads sequences of length l (line 9) and extracts features using the context window size ω (line 10). The extracted features, gradient, current learning rate and clustering cost coefficient determine the update to model weights by the optimizer (line 11). In the implementation we utilized a variant of stochastic gradient descent – the *Adam* optimizer [12]. We approximate the first cost term with noise contrastive estimation to make the gradient descent tractable, drawing k noise samples for each positive sample. If the node sampling is done by first-order random walks the runtime complexity of this procedure will be $\mathcal{O}((\omega \cdot k + |C|) \cdot l \cdot d \cdot |V| \cdot N)$ while *DeepWalk* with noise contrastive estimation has a $\mathcal{O}(\omega \cdot k \cdot l \cdot d \cdot |V| \cdot N)$ runtime complexity.

C. Smoothness Regularization for coherent community detection

We have seen in Subsection III-A that there is a tension between what the clustering objective considers to be clusters and what the real communities are in the underlying social network. We can incorporate additional knowledge of social network communities using a machine learning technique called regularization.

We observe that social networks have natural local properties such as homophily, strong ties between members of a community, etc. Thus, we can incorporate such social network-specific properties in the form of regularization to find more natural embeddings and clusters.

This regularization effect can be achieved by adding a term Λ to the loss function:

$$\Lambda = \lambda \cdot \sum_{(v,u) \in E_S} w_{(v,u)} \cdot \|f(v) - f(u)\|_2 , \tag{10}$$

where the weight function w determines the *social network cost* of the embedding with respect to properties of the edges traversed in the sampling. We use the neighborhood overlap of an edge – defined as the fraction of neighbors common to two nodes of the edge relative to the union of the two neighbor sets[2]. In experiments on real data, neighborhood overlap is known to be a strong indicator of the strength of relation between members of a social network [28]. Thus, by treating neighborhood overlap as the weight $w_{v,u}$ of edge (v, u), we can get effective social network clustering, which is confirmed by experiments in the next section. The coeffeicient λ lets us tune the contribution of the social network cost in the embedding process. In experiments, the regularized version of the algorithms is found to be more robust to changes in hyperparameters.

The effect of the regularization can be understood intuitively through an example. For this exposition, let us consider matrix representations of the social network describing closeness of nodes. In fact, other skip-gram style learning processes like [8], [9] are known to approximate the factorization of

[2]Neighbor sets $N(a)$ and $N(b)$ of nodes a and b, the neighborhood overlap of (a, b) is defined as the Jaccard similarity $\frac{N(a) \cap N(b)}{N(a) \cup N(b)}$.

a similarity matrix M such as [26]:

$$M_{u,v} = \log\left(\frac{\mathrm{vol}(G)}{\omega}\sum_{r=1}^{\omega}\frac{\sum_{P\in\mathcal{P}^r_{v,u}}\prod_{a\in P\setminus\{v\}}\frac{1}{\deg(a)}}{\deg(v)}\right) - \log(k)$$

where $P^r_{v,u}$ is the set of paths going from v to u with length r. Elements of the target matrix M grow with number of paths of length at most ω between the corresponding nodes. Thus M is intended to represent level of connectivity between nodes in terms of a raw graph feature like number of paths.

The barbell graph in Figure 3a is a typical example with an obvious community structure we can use to analyze the matter. The optimization procedure used by Deepwalk [8] aims to converge to a target matrix $M_{u,v}$ shown in Figure 3b. Observe that this matrix has fuzzy edges around the communities of the graph, showing a degree of uncertainty. An actual approximation by running the Deepwalk is shown in Figure 3c, which naturally incorporates further uncertainty due to sampling. A much more clear output with sharp communities can be obtained by applying a regularized optimization. This can be seen in Figure 3d.

TABLE I
STATISTICS OF SOCIAL NETWORKS USED IN THE PAPER.

Source	Dataset	|V|	Density	Transitivity
Facebook	Politicians	5,908	0.0024	0.3011
	Companies	14,113	0.0005	0.1532
	Athletes	13,866	0.0009	0.1292
	Media	27,917	0.0005	0.1140
	Celebrities	11,565	0.0010	0.1666
	Artists	50,515	0.0006	0.1140
	Government	7,057	0.0036	0.2238
	TV Shows	3,892	0.0023	0.5906
Deezer	Croatia	54,573	0.0004	0.1146
	Hungary	47,538	0.0002	0.0929
	Romania	41,773	0.0001	0.0752

IV. EXPERIMENTAL EVALUATION

In this section we evaluate the cluster quality obtained by the *GEMSEC* variants, their scalability, robustness and predictive performance on a downstream supervised task. Results show that *GEMSEC* outperforms or is at par with existing methods in all measures.

A. Datasets

For the evaluation of *GEMSEC* real-world social network datasets are used which we collected from public APIs specifically for this work. Table I shows these social networks have a variety of size, density, and level of clustering. We used graphs from two sources:

- *Facebook page networks:* These graphs represent mutual like networks among verified Facebook pages – the types of sites included TV shows, politicians, athletes, and artists among others.
- *Deezer user-user friendship networks:* We collected friendship networks from the music streaming site Deezer and included 3 European countries (Croatia, Hungary, and Romania). For each user, we curated the list of genres loved based on the songs liked by the user.

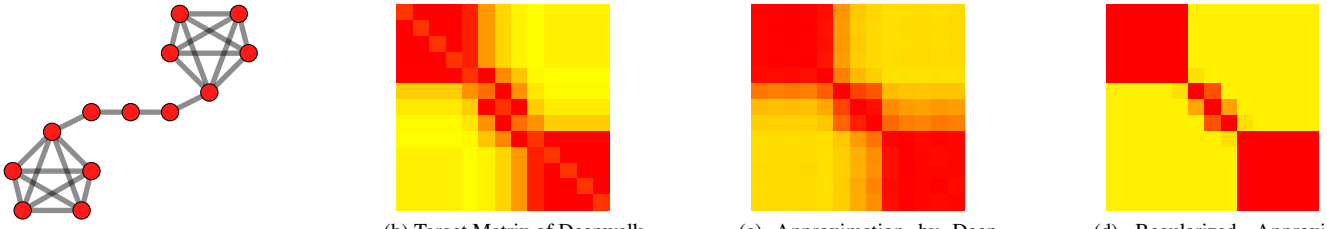
B. Standard parameter settings

A fixed standard parameter setting is used our experiments, and we indicate any deviations. Models using first order random walk sampling strategy are referenced as *GEMSEC* and *Smooth GEMSEC*, second order random walk variants are named as *GEMSEC*$_2$ and *Smooth GEMSEC*$_2$. Random walks with length 80 are used and 5 truncated random walks per source node were used. Second-order random walk control hyperparameters [9] *return* and *in-out* were chosen from $\{2^{-2}, 2^{-1}, 1, 2, 4\}$. A window size of 5 is used for features. Each embedding has 16 dimensions and we extract 20 cluster centers. A parameter sweep over hyperparameters was used to obtain the highest average modularity. Initial learning rate values are chosen from $\{10^{-2}, 5\cdot10^{-3}, 10^{-3}\}$ and the final learning rate is chosen from $\{10^{-3}, 5\cdot10^{-4}, 10^{-4}\}$. Noise contrastive estimation uses 10 negative examples. The initial clustering cost coefficient is chosen from $\{10^{-1}, 10^{-2}, 10^{-3}\}$. The smoothness regularization term's hyperparameter is 0.0625 and Jaccard's coefficient is the penalty weight.

C. Cluster Quality

Using Facebook page networks we evaluate the clustering performance. Cluster quality is evaluated by modularity – we assume that a node belongs to a single community. Our results are summarized in Table II based on 10 experimental repetitions and errors in parentheses correspond to two standard deviations. The baselines use the hyperparameters from the respective papers. We used 16-dimensional embeddings throughout. The embeddings obtained with non-community-aware methods were clustered after the embedding by k-means clustering to extract 20 cluster centers. Specifically, comparisons are made with:

1) *Overlap Factorization* [29]: Factorizes the neighborhood overlap matrix to create features.
2) *DeepWalk* [8]: Approximates the sum of the adjacency matrix powers with first order random walks and implicitly factorizes it.
3) *LINE* [16]: Implicitly factorizes the sum of the first two powers for the normalized adjacency matrix and the resulting node representation vectors are concatenated together to form a multi-scale representation.
4) *Node2vec* [9]: Factorizes a neighbourhood matrix obtained with second order random walks. The *in-out* and *return* parameters of the second-order random walks were chosen from the $\{2^{-2}, 2^{-1}, 1, 2, 4\}$ set to maximize modularity.
5) *Walklets* [17]: Approximates with first order random walks each adjacency matrix power individually and implicitly factorizes the target matrix. These embeddings



(a) The graph (b) Target Matrix of Deepwalk. (c) Approximation by Deepwalk. (d) Regularized Approximation

Fig. 3. An example Barbell graph with the corresponding target matrix factorized (window size of 3) by *DeepWalk* [26] and the reconstructed target matrices obtained with standard *DeepWalk* and *Smooth DeepWalk*. Regularized optimization produces more well defined communities. While the standard *DeepWalk* model has less well defined clusters.

TABLE II

MEAN MODULARITY OF CLUSTERINGS ON THE FACEBOOK DATASETS. EACH EMBEDDING EXPERIMENT WAS REPEATED TEN TIMES. ERRORS IN THE PARENTHESES CORRESPOND TO TWO STANDARD DEVIATIONS. IN TERMS OF MODULARITY *Smooth GEMSEC*$_2$ OUTPERFORMS THE BASELINES.

	Politicians	**Companies**	**Athletes**	**Media**	**Celebrities**	**Artists**	**Government**	**TV Shows**
Overlap Factorization	0.810 (±0.008)	0.553 (±0.010)	0.601 (±0.020)	0.471 (±0.016)	0.551 (±0.01)	0.474 (±0.018)	0.608 (±0.024)	0.786 (±0.008)
DeepWalk	0.840 (±0.015)	0.637 (±0.012)	0.649 (±0.012)	0.481 (±0.022)	0.631 (±0.011)	0.508 (±0.029)	0.686 (±0.024)	0.811 (±0.005)
LINE	0.841 (±0.014)	0.651 (±0.009)	0.665 (±0.007)	0.558 (±0.012)	0.642 (±0.010)	0.557 (±0.014)	0.690 (±0.017)	0.813 (±0.010)
Node2Vec	0.846 (±0.012)	0.664 (±0.008)	0.669 (±0.007)	0.565 (±0.011)	0.643 (±0.013)	0.560 (±0.010)	0.692 (±0.017)	0.827 (±0.016)
Walklets	0.843 (±0.014)	0.655 (±0.012)	0.664 (±0.007)	0.562 (±0.009)	0.621 (±0.043)	0.548 (±0.016)	0.689 (±0.019)	0.819 (±0.015)
ComE	0.830 (±0.008)	0.654 (±0.005)	0.665 (±0.007)	**0.573** (±0.005)	0.635 (±0.010)	0.560 (±0.011)	0.696 (±0.010)	0.806 (±0.011)
M-NMF	0.816 (±0.014)	0.646 (±0.007)	0.655 (±0.008)	0.561 (±0.004)	0.628 (±0.006)	0.535 (±0.021)	0.668 (±0.011)	0.813 (±0.008)
DANMF	0.810 (±0.020)	0.648 (±0.005)	0.650 (±0.009)	0.560 (±0.006)	0.628 (±0.011)	0.532 (±0.019)	0.673 (±0.015)	0.812 (±0.014)
Smooth DeepWalk	0.849 (±0.017)	0.667 (±0.007)	0.669 (±0.007)	0.541 (±0.006)	0.643 (±0.008)	0.523 (±0.020)	0.707 (±0.008)	0.835 (±0.008)
GEMSEC	0.851 (±0.009)	0.662 (±0.013)	0.674 (±0.009)	0.536 (±0.011)	0.636 (±0.014)	0.528 (±0.020)	0.705 (±0.020)	0.833 (±0.010)
Smooth GEMSEC	0.855 (±0.006)	0.683 (±0.009)	**0.692** (±0.009)	0.567 (±0.009)	**0.649** (±0.008)	0.559 (±0.011)	0.710 (±0.008)	0.841 (±0.004)
GEMSEC$_2$	0.852 (±0.010)	0.667 (±0.008)	0.683 (±0.008)	0.551 (±0.008)	0.638 (±0.009)	**0.562** (±0.020)	**0.712** (±0.010)	0.838 (±0.010)
Smooth GEMSEC$_2$	**0.859** (±0.006)	**0.684** (±0.009)	**0.692** (±0.007)	0.571 (±0.010)	**0.649** (±0.011)	**0.562** (±0.017)	**0.712** (±0.010)	**0.847** (±0.006)

are concatenated to form a multi-scale representation of nodes.

6) *ComE* [13]: Uses a Gaussian mixture model to learn an embedding and clustering jointly using random walk features.
7) *M-NMF* [14]: Factorizes a matrix which is a weighted sum of the first two proximity matrices with a modularity based regularization constraint.
8) *DANMF* [15]: Decomposes a weighted sum of the first two proximity matrices hierarchically to obtain cluster memberships with an autoencoder-like non-negative matrix factorization model.

Smooth GEMSEC, *GEMSEC*$_2$ and *Smooth GEMSEC*$_2$ consistently outperform the neighborhood conserving node embedding methods and the competing community aware methods. The relative advantage of *Smooth GEMSEC*$_2$ over the benchmarks is highest on the Athletes dataset as the clustering's modularity is 3.44% higher than the best performing baseline. It is the worst on the Media dataset with a disadvantage of 0.35% compared to the strongest baseline. Use of smoothness regularization has sometimes non-significant, but definitely positive effect on the clustering performance of *Deepwalk*, *GEMSEC* and *GEMSEC*$_2$.

D. Sensitivity Analysis for hyperparameters

We tested the effect of hyperparameter changes to clustering performance. The Politicians Facebook graph is embedded with the standard parameter settings while the initial and final learning rates are set to be 10^{-2} and $5 \cdot 10^{-3}$ respectively, the clustering cost coefficient is 0.1 and we perturb certain hyperparameters. The second-order random walks used *in-out* and *return* parameters of 4. In Figure 4 each data point represents the mean modularity calculated from 10 experiments. Based on the experimental results we make two observations. First, *GEMSEC* model variants give high-quality clusters for a wide range of parameter settings. Second, introducing smoothness regularization makes *GEMSEC* models more robust to hyperparameter changes. This is particularly apparent across varying the number of clusters. The length of truncated random walks and the number of random walks per source node above a certain threshold has only a marginal effect on the community detection performance.

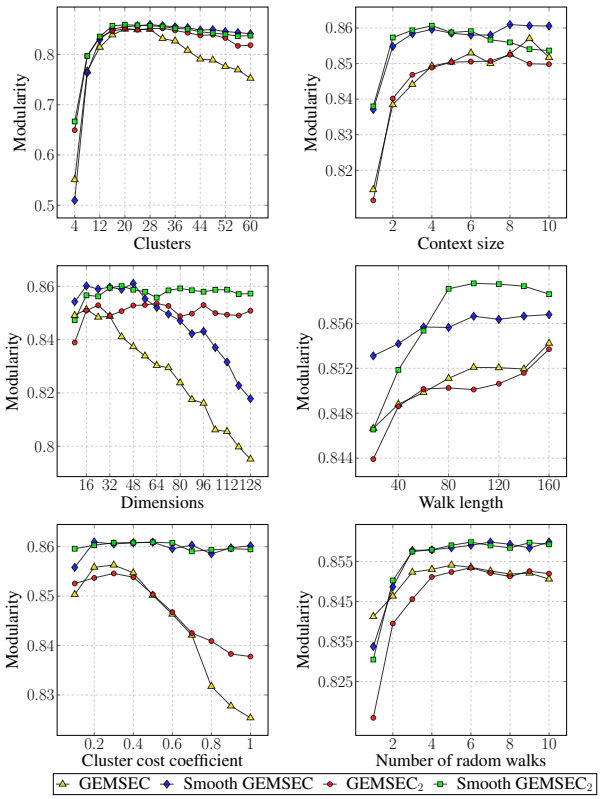


Fig. 4. Sensitivity of cluster quality to parameter changes measured by modularity.

E. Music Genre Recommendation

Node embeddings are often used for extracting features of nodes for downstream predictive tasks. In order to investigate this, we use social networks of Deezer users collected from European countries. We predict the genres (out of 84) of music liked by people. Following the embedding, we used logistic regression with ℓ_2 regularization to predict each of the labels and 90% of the nodes were randomly selected for training. We evaluated the performance of the remaining users. Numbers reported in Table III are F_1 scores calculated from 10 experimental repetitions. *GEMSEC*$_2$ significantly outperforms the other methods on all three countries' datasets. The performance advantage varies between 3.03% and 4.95%. We also see that *Smooth GEMSEC*$_2$ has lower accuracy, but it is able to outperform *DeepWalk, LINE, Node2Vec, Walklets, ComE, M-NMF* and *DANMF* on all datasets.

F. Scalability and computational efficiency

To create graphs of various sizes, we used the Erdos-Renyi model and with an average degree of 20. Figure 5 shows the log of mean runtime against the log of the number of nodes. Most importantly, we can conclude that doubling the size of the graph doubles the time needed for optimizing *GEMSEC*, thus the growth is linear. We also observe that embedding algorithms that incorporate clustering have a higher cost, and regularization also produces a higher cost, but similar growth.

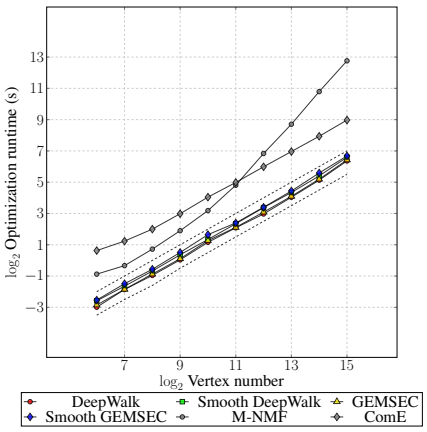


Fig. 5. Sensitivity of optimization runtime to graph size measured by seconds. The dashed lines are linear references.

V. Conclusions

We described *GEMSEC* – a novel algorithm that learns a node embedding and a clustering of nodes jointly. It extends existing embedding modes. We showed that smoothness regularization is used to incorporate social network properties and produce natural embedding and clustering. We presented new social datasets, and experimentally, our methods outperform a number of strong community aware node embedding baselines.

VI. Acknowledgements

Benedek Rozemberczki and Ryan Davies were supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1).

References

[1] T. Van Laarhoven and E. Marchiori, "Robust community detection methods with resolution parameter for complex detection in protein protein interaction networks," *Pattern Recognition in Bioinformatics*, pp. 1–13, 2012.

[2] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: Membership, growth, and evolution," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 44–54.

[3] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos, "Community detection in social media," *Data Mining and Knowledge Discovery*, vol. 24, no. 3, pp. 515–554, 2012.

[4] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2014.

[5] P. Pascal and M. Latapy, *In International Symposium on Computer and Information Sciences*. Springer Berlin Heidelberg, 2005, ch. Computing Communities in Large Networks Using Random Walks., pp. 284–293.

[6] S. Gregory, "Finding overlapping communities in networks by label propagation," *New Journal of Physics*, vol. 12, no. 10, p. 103018, 2010.

[7] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *arXiv preprint arXiv:1705.02801*, 2017.

[8] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations." in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.*, 2014.

[9] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 855–864.

TABLE III

MULTI-LABEL NODE CLASSIFICATION PERFORMANCE OF THE EMBEDDING EXTRACTED FEATURES ON THE DEEZER GENRE LIKES DATASETS. PERFORMANCE IS MEASURED BY AVERAGE F_1 SCORE VALUES. MODELS WERE TRAINED ON 90% OF THE DATA AND EVALUATED ON THE REMAINING 10%. ERRORS IN THE PARENTHESES CORRESPOND TO TWO STANDARD DEVIATIONS. *GEMSEC* MODELS CONSISTENTLY HAVE GOOD PERFORMANCE.

	CROATIA			**HUNGARY**			**ROMANIA**		
	Micro	**Macro**	**Weighted**	**Micro**	**Macro**	**Weighted**	**Micro**	**Macro**	**Weighted**
Overlap Factorization	0.319 (±0.017)	0.026 (±0.002)	0.208 (±0.010)	0.361 (±0.007)	0.029 (±0.001)	0.227 (±0.006)	0.275 (±0.025)	0.020 (±0.003)	0.167 (±0.017)
DeepWalk	0.321 (±0.006)	0.026 (±0.002)	0.207 (±0.004)	0.361 (±0.004)	0.029 (±0.002)	0.228 (±0.002)	0.307 (±0.008)	0.023 (±0.002)	0.186 (±0.006)
LINE	0.331 (±0.013)	0.028 (±0.002)	0.212 (±0.010)	0.374 (±0.007)	0.033 (±0.002)	0.250 (±0.005)	0.332 (±0.007)	0.028 (±0.002)	0.212 (±0.006)
Node2Vec	0.348 (±0.012)	0.032 (±0.003)	0.235 (±0.010)	0.393 (±0.008)	0.037 (±0.002)	0.267 (±0.011)	0.346 (±0.008)	0.031 (±0.002)	0.229 (±0.008)
Walklets	0.363 (±0.013)	0.043 (±0.003)	0.270 (±0.012)	0.397 (±0.007)	0.051 (±0.001)	0.307 (±0.006)	0.361 (±0.011)	**0.050** (±0.005)	0.281 (±0.012)
ComE	0.326 (±0.012)	0.028 (±0.002)	0.217 (±0.009)	0.363 (±0.010)	0.033 (±0.001)	0.246 (±0.007)	0.323 (±0.008)	0.028 (±0.001)	0.212 (±0.006)
M-NMF	0.336 (±0.005)	0.028 (±0.001)	0.217 (±0.003)	0.369 (±0.015)	0.032 (±0.002)	0.239 (±0.011)	0.330 (±0.016)	0.028 (±0.002)	0.209 (±0.013)
DANMF	0.340 (±0.007)	0.027 (±0.002)	0.210 (±0.002)	0.365 (±0.011)	0.031 (±0.002)	0.242 (±0.008)	0.335 (±0.009)	0.029 (±0.002)	0.210 (±0.012)
Smooth DeepWalk	0.329 (±0.006)	0.028 (±0.002)	0.215 (±0.006)	0.375 (±0.006)	0.032 (±0.002)	0.244 (±0.004)	0.321 (±0.008)	0.026 (±0.002)	0.204 (±0.006)
GEMSEC	0.328 (±0.006)	0.027 (±0.002)	0.212 (±0.004)	0.377 (±0.004)	0.032 (±0.002)	0.244 (±0.004)	0.332 (±0.008)	0.028 (±0.002)	0.213 (±0.006)
Smooth GEMSEC	0.333 (±0.006)	0.028 (±0.002)	0.215 (±0.004)	0.379 (±0.006)	0.034 (±0.002)	0.250 (±0.004)	0.334 (±0.008)	0.029 (±0.002)	0.215 (±0.006)
GEMSEC$_2$	**0.381** (±0.007)	**0.046** (±0.003)	**0.287** (±0.005)	0.407 (±0.005)	0.050 (±0.003)	0.310 (±0.007)	**0.378** (±0.009)	0.049 (±0.003)	**0.289** (±0.007)
Smooth GEMSEC$_2$	0.373 (±0.005)	0.044 (±0.002)	0.276 (±0.006)	**0.409** (±0.004)	**0.053** (±0.002)	**0.314** (±0.006)	0.376 (±0.008)	0.049 (±0.003)	0.287 (±0.007)

[10] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[12] J. B. Diederik P. Kingma, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[13] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 377–386.

[14] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding." in *AAAI*, 2017, pp. 203–209.

[15] F. Ye, C. Chen, and Z. Zheng, "Deep autoencoder-like nonnegative matrix factorization for community detection," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 1393–1402.

[16] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 1067–1077.

[17] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena, "Don't walk, skip!: Online learning of multi-scale network embeddings," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, 2017, pp. 258–265.

[18] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," *Journal of Computer and System Sciences*, vol. 69, no. 3, pp. 485–497, 2004.

[19] J. Matoušek, *Lectures on discrete geometry*. Springer, 2002, vol. 108.

[20] X. Yu, X. Ban, W. Zeng, R. Sarkar, X. Gu, and J. Gao, "Spherical representation and polyhedron routing for load balancing in wireless sensor networks," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 621–625.

[21] K. Huang, C.-C. Ni, R. Sarkar, J. Gao, and J. S. Mitchell, "Bounded stretch geographic homotopic routing in sensor networks," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 979–987.

[22] R. Sarkar, "Low distortion delaunay embedding of trees in hyperbolic plane," in *International Symposium on Graph Drawing*. Springer, 2011, pp. 355–366.

[23] C. De Sa, A. Gu, C. Ré, and F. Sala, "Representation tradeoffs for hyperbolic embeddings," *Proceedings of machine learning research*, vol. 80, p. 4460, 2018.

[24] W. Zeng, R. Sarkar, F. Luo, X. Gu, and J. Gao, "Resilient routing for sensor networks using hyperbolic embedding of universal covering space," in *2010 Proceedings IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[25] B. Rozemberczki and R. Sarkar, "Fast sequence-based embedding with diffusion graphs," in *International Workshop on Complex Networks*. Springer, 2018, pp. 99–107.

[26] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 2018, pp. 459–467.

[27] M. Gutmann and A. Hyvarinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 297–304.

[28] J.-P. Onnela, J. Saramäki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A.-L. Barabási, "Structure and tie strengths in mobile communication networks," *Proceedings of the national academy of sciences*, vol. 104, no. 18, pp. 7332–7336, 2007.

[29] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 37–48.