Fast Sequence Based Embedding with Diffusion Graphs

Fast Sequence Based Embedding with Diffusion Graphs

Benedek Rozemberczki and Rik Sarkar

School of Informatics, University of Edinburgh, U.K.
benedek.rozemberczki@ed.ac.uk
rsarkar@inf.ed.ac.uk

Abstract. A graph embedding is a representation of the vertices of a graph in a low dimensional space, which approximately preserves properties such as distances between nodes. Vertex sequence based embedding procedures use features extracted from linear sequences of vertices to create embeddings using a neural network. In this paper, we propose diffusion graphs as a method to rapidly generate vertex sequences for network embedding. Its computational efficiency is superior to previous methods due to simpler sequence generation, and it produces more accurate results. In experiments, we found that the performance relative to other methods improves with increasing edge density in the graph. In a community detection task, clustering nodes in the embedding space produces better results compared to other sequence based embedding methods.

1 Introduction

Embedding graphs into a low dimensional Euclidean spaces is a way of simplifying the graph information by associating each node with a point in the space. Thus, various methods of graph embedding have been developed and applied to different domains, such as visualisation [10], community and cluster identification [27], localisation of wireless devices [22], network routing [21], construction of approximate distance oracle [12] etc. Graph embeddings usually aim to preserve proximity – nearby nodes on the graph should have similar coordinates – in addition to properties specific to the application.

In recent years, sequence based graph embedding methods have been developed as a way of generating Euclidean representations using sequence of vertices obtained from random walks. These methods are inspired by Word2Vec – a method to embed words into Euclidean space based on sequences in which they occur. Word2vec takes short sequences of words from a document and uses them to train a neural network; in the process it obtains an embedding for the words. The embedding space acts as an abstract latent space of *features*, and usually places two words close if they frequently occur nearby in the sequences [16,15]. Sequence based graph embedding methods on the other hand obtain their vertex sequences by random walk on graphs and then apply analogous neural network methods for the embedding. The random walk has the advantage that it obtains

a view of the neighborhood, without having to compute and store complete neighborhoods, which can be expensive in a large graph with many high degree vertices.

However, random walks are inefficient for generating proximity statistics. They are known to spread slowly, and revisit a vertex many times producing redundant information [3]. As a result, they require many steps or many restarts to cover the neighborhood of a node. Methods like Node2vec [9] try to bias the walks away from recently visited nodes, but in the process they incur a cost due to the complexity of modifying transition probabilities with each step. We instead use a diffusion process that samples a subgraph of the neighborhood, from which several walks can be generated more efficiently.

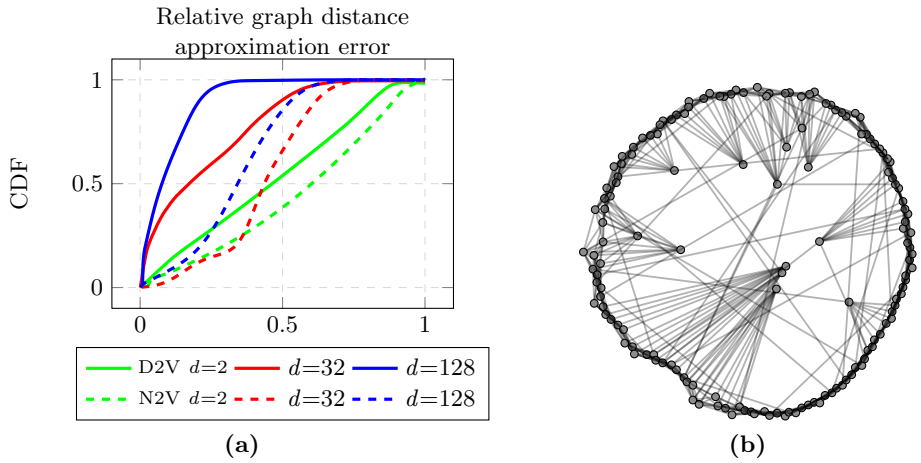

Fig. 1: (a) Cumulative distribution of the shortest path distance approximation error on the PPI network [5] ($|V|$=3,890) for embedding dimension d. The distortion for our method (D2V) is much smaller than state of the art (N2V). The distortion error for nodes u and v is defined as $e_{u,v} = |d(u,v) - \gamma \cdot \|\mathbf{X}_v - \mathbf{X}_u\|| \, / d(u,v)$. Embeddings were created with parameter settings such that $n = 10$, $\widehat{w} = 10$, $\alpha = 0.025$, $k = 1, l = 40$ (D2V) and $l = 80$ (N2V). The best inout and return parameters of N2V were chosen with grid search over $\{0.25, 0.5, 1, 2, 4\}$. **(b)** Visualization of a Watts-Strogatz graph [25] with our embedding procedure.

Our Contributions. In our method, we extract a subgraph of the neighborhood of a node using a diffusion-like process, and call it a *diffusion graph*. On this subgraph, we compute an Euler tour to use as a sequence. By covering all adjacencies in the graph, the Euler tour contains a more complete view of the local neighborhood than random walks. We refer to this sequence generating method as *Diff2Vec* (D2V). The sequences generated by Diff2Vec are then used to train a neural network with one hidden layer containing d neurons for d-dimensional embedding. The input weights of the neurons determine the embedding of the nodes. We found in experiments that this system has several desirable properties.

Due to its better coverage of neighborhoods, Diff2Vec can operate with smaller neighborhood samples. As a result, it is more efficient than existing methods. In our experiments with a basic implementation, it turned out to be

several times faster. In particular, it scales better with increasing density (vertex degrees) of graphs.

Our experiments also show that the embedding preserves graph distances to a high accuracy. On experiments of community detection, we found that clustering applied to the embedding produces communities of high quality – verified by the high modularity of the clusters. The quality of clusters is robust to hyperparameter changes including parameters such as the embedding dimension and number of diffusion graphs generated per node.

The remainder of the paper is structured as follows. In Section 2 we overview related literature. The theoretical background on sequence based graph embedding is discussed in Section 3. Our sampling procedure is introduced in Section 4. Experimental results are analyzed in Section 5 and the paper concludes with Section 6.

2 Related Works

In this section we discuss the most well known graph embedding techniques and recent developments regarding them. The well known embedding techniques use a matrix that describes the graph and factorize it in order to create the embedding of the network. One can factorize the adjacency, neighbourhood overlap or Laplacian matrices. Based on the properties of the matrix either eigenvalue decomposition or some variant of stochastic gradient descent is used to obtain the graph embedding. These embedding methods all have a weakness, namely that they are computationally expensive. We refer the reader to the recent survey in [8] for a broader overview of graph embedding, and focus here on relevant neural network based embeddings.

Sequence based embedding. Node sequence based graph embedding methods were inspired by word embedding procedures, specifically by the *skip-gram* model [15,16]. The generation of node sequence based graph embeddings consists of three phases. First, the algorithm creates vertex sequences - usually by a random process. Second, features that are extracted from the synthetic sequences describe the approximated proximities of nodes. Finally, the embedding itself is learned using the extracted node specific features with a neural network which has a single hidden layer. Sequence based embedding originates from the *DeepWalk* model [18] which uses random walks to generate node sequences. This approach was improved upon by *Node2Vec* (henceforth N2V) [9] which uses second-order random walks to generate the vertex sequences. Second-order random walks alternate between depth-first and breadth-first search on the graph in a random, but somewhat controlled way. In this attempt to have greater control on random walks, N2V introduces parameters that affect the embedding quality and are hard to optimize.

Other neural network based approaches. Other graph embedding methods that use neural networks are all based on the of encoding a matrix representation of the graph. At the same time they maintain first and higher order proximities or distances. The matrix representation used for creating the embedding can be

the adjacency or the random walk transmission probability matrix [24,4]. Recent developments in graph signal processing allow the use of generic vertex features in addition when an embedding is created [23]. Building on this progress of graph signal processing graph convolutional deep neural networks were introduced that create embeddings that use generic vertex features [11].

3 Feature extraction and neural network embedding

Feature extraction. In this section, we discuss the technique of generating embeddings when given some sequences of vertices. The sequence generation will be taken up in the following section.

We start with extracting features called hitting frequency vectors – denoting frequencies with which vertices occur near each other. The graph is denoted by $\mathcal{G}(V, E)$. The set of vertices is V and the edge set is E. We assume that the graph is undirected and unweighted. Let us consider an example to see how an embedding is generated.

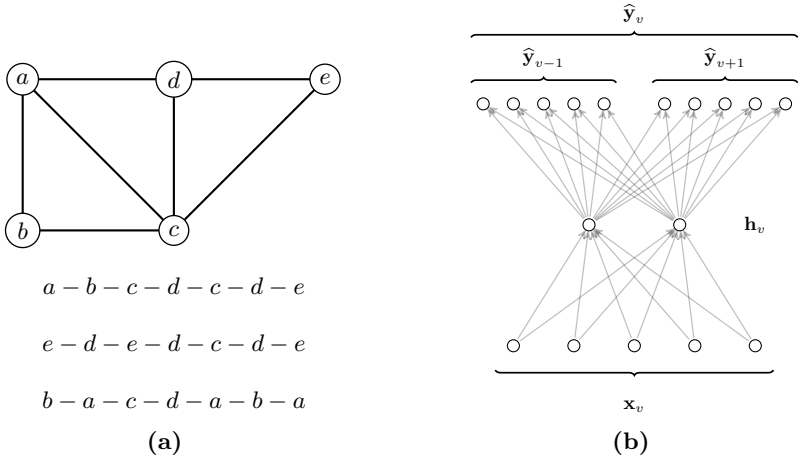

Fig. 2: (a) Graph with linear vertex sequences. The three vertex sequences listed are used for feature extraction in our example. **(b)** Architecture of the example neural network.

Consider the example in Figure 2(a). The vertex set contains nodes a, b, c, d, e and nodes are indexed respectively from 1 to 5, and suppose we are given the 3 node sequences in the figure. To generate features from the sequences we choose a sliding window size denoted by $\widehat{w}$ which limits the maximal graph proximity among nodes that we are going to approximate. In this case we choose $\widehat{w} = 1$. We calculate the co-occurrence frequencies for node c as follows – we count how many times other nodes appeared at given positions before and after c limited by the window's size. In this toy example it means positions at maximal 1 step before or after c in the sequence. Counts at different positions are stored in separate vectors for each node. The resulting frequency vectors are as follows: $\mathbf{y}_{c,-1} = \begin{bmatrix}1\ 1\ 0\ 2\ 0\end{bmatrix}$ and $\mathbf{y}_{c,+1} = \begin{bmatrix}0\ 0\ 0\ 4\ 0\end{bmatrix}$. Components of the vectors can be interpreted as noisy proximity statistics in the graph. The idea is that nearby nodes will have higher

values in each-other's vectors. We concatenate these vectors to form a vector of $2 \cdot \widehat{w} \cdot |V|$ components and call it the hitting frequency vector $\mathbf{y}_v$ of a node v. We construct such a hitting frequency vector for each node from the given sequences.

Learning an embedding from the features. For each vertex $v \in V$, we wish to compute a coordinate in $\mathbb{R}^d$. The set of hitting frequency vectors is a representation of the graph in $\mathbb{R}^{|V| \times 2 \cdot \widehat{w} \cdot |V|}$, which we have to reduce to a $\mathbb{R}^{|V| \times d}$ space.

We write as $\mathbf{x}_v$ the indicator (sometimes called hot-one) vector for v, which has $|V|$ elements, all of which are zero, except the element at index of v, which is set to 1.

A schematic of the neural network architecture is in Figure 2(b). The neural network has d hidden neurons, each with $|V|$ inputs and $2 \cdot \widehat{w} \cdot |V|$ outputs. The incoming and outgoing weight matrices of the hidden neurons are written as $\mathbf{W}_{in}$ and $\mathbf{W}_{out}$. To train the neural network, the training algorithm uses input output pairs of the form $(\mathbf{x}_v, \mathbf{y}_v)$ corresponding to each vertex v. Thus, the neural network learns to associate with each vertex, an output that is its hitting frequency vector. After the training, the incoming weight matrix $\mathbf{W}_{in}$ (of dimension $d \times |V|$) gives the d dimensional embedding of the vertices.

In more detail, the neural network is designed as follows. For any v, we can define

$$\mathbf{h}_v = \sigma(\mathbf{W}_{in} \cdot \mathbf{x}_v + \mathbf{b}_{in}) \tag{1}$$

where in our case $\sigma(\mathbf{W}_{in} \cdot \mathbf{x}_v + \mathbf{b}_{in}) = \mathbf{W}_{in} \cdot \mathbf{x}_v + \mathbf{b}_{in}$. And $\mathbf{b}_{in}$ is the bias vector. The output of the network is given by Equation 2, with the predicted hitting frequency vector $\widehat{\mathbf{y}}_v$ on the left hand side. The activation function Φ in the final layer is chosen as the hierarchical softmax function.

$$\widehat{\mathbf{y}}_v = \Phi(\mathbf{W}_{out} \cdot \mathbf{h}_v + \mathbf{b}_{out}) \tag{2}$$

The goal is to approximate $\mathbf{y}_v$. We define the approximation error as the loss function $\mathcal{L}$ – the multinomial log loss of $\mathbf{y}_v$ and $\widehat{\mathbf{y}}_v$, written as $\mathcal{L}(\mathbf{y}_v, \widehat{\mathbf{y}}_v) = -\mathbf{y}_v \cdot \log(\widehat{\mathbf{y}}_v)$. Over all vertices, our minimization objective is given as:

$$\min \quad \sum_{v \in V} \mathcal{L}\left(\mathbf{y}_v, \Phi(\mathbf{W}_{out} \cdot \mathbf{h}_v + \mathbf{b}_{out})\right) \tag{3}$$

The weight matrix $\mathbf{W}_{in}$ is the embedding itself – for each v in V we have a d dimensional representation in a latent space. The weight matrix is used to approximately reconstruct the hitting frequencies of a node. If two nodes have similar hitting frequency vectors, meaning that their proximity is high, they will also have a similar latent space representation. Our goal is the efficient and scalable learning of the embedding so we use asynchronous gradient descent (ASGD, [20]). Analogous to previous works [18,9], we used hierarchical softmax activation, with which the computational complexity of a training epoch (while we decrease the learning rate from starting value to zero) is $\mathcal{O}(|V| \log(|V|))$. We refer to the embedding as $\mathbf{X}$, and the embedding of node v is noted by $\mathbf{X}_v$.

4 Sequence generation algorithm and design

To generate sequences in the neighborhood of a node, we first compute a *diffusion graph*, and then use that to compute vertex sequences.

Diffusion graph generation. We emulate a simple diffusion-like random process starting from a vertex v to sample a subgraph of l vertices near v. The diffusion graph $\widetilde{\mathcal{G}}$ is initialized with $\{v\}$. Next, at each step, we sample a random node u from $\widetilde{\mathcal{G}}$ and from the neighbors of u in the original graph $\mathcal{G}$, we select w. We add w to the set of vertices in $\widetilde{\mathcal{G}}$, and add the edge (u, w) to $\widetilde{\mathcal{G}}$. This process is repeated until $\widetilde{\mathcal{G}}$ has l nodes. An example of a diffusion graph is depicted on Figure 3.

Data: $\mathcal{G}$ – Graph object.
l – Number of nodes sampled.
v – Starting node .
Result: P – Eulerian sequence from v.

$V_{\widetilde{\mathcal{G}}} \leftarrow \{v\}$
while $|V_{\widetilde{\mathcal{G}}}| < l$ **do**
  $w \leftarrow$ Random Sample($V_{\widetilde{\mathcal{G}}}$)
  $u \leftarrow$ Random Sample($N_{\mathcal{G}}(w)$)
  if $u \notin V_{\widetilde{\mathcal{G}}}$ **then**
    $V_{\widetilde{\mathcal{G}}} \leftarrow V_{\widetilde{\mathcal{G}}} \cup \{u\}$
    $E_{\widetilde{\mathcal{G}}} \leftarrow E_{\widetilde{\mathcal{G}}} \cup \{(u, w)\}$
  end
end
$\widetilde{\mathcal{G}} \leftarrow$ Duplicate Edges($\widetilde{\mathcal{G}}$)
$P \leftarrow$ Random Eulerian Circuit($\widetilde{\mathcal{G}}, v$)

Algorithm 1: Graph sampling

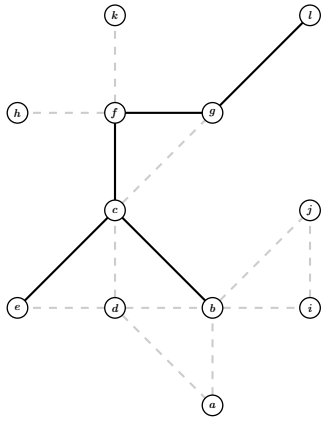

Fig. 3: Diffusion graph example

Node sequence sampling. To generate sequences from the subgraph $\widetilde{\mathcal{G}}$, we take the following approach. We convert $\widetilde{\mathcal{G}}$ into a multigraph by doubling each edge into two edges. A connected graph where every node has an even degree is Eulerian[1], and the Euler walk is easy to find [26]. We use this method to find the Euler walk and use that as a vertex sequence. Observe that this diffusion graph sampling and sequence generation can be performed in parallel across many machines, since each diffusion graph can be generated independent of others (see schematic on Figure 4). The generated sequences are then used to produce graph embedding using neural networks as seen in the previous section. Note that an Euler walk has the nice property that it captures every adjacency relation in the subgraph into a linear sequence using asymptotically optimal space. This property then helps our method perform better both in the sense of efficiency and quality of results.

[1] Contains a walk that passes through every edge exactly once

Data: $\mathcal{G}$ – Graph embedded.
p – Sequence samples per node.
l – Number of nodes per sample.
d – Dimension of embedding.
k – Number of epochs.
$\widehat{w}$ – Size of sliding window.
α – Learning rate.
Result: $\mathbf{X}$ – Embedding of graph $\mathcal{G}$.

$\mathcal{G}_1, \dots \mathcal{G}_S \leftarrow$ Component Extraction($\mathcal{G}$)
Samples $\leftarrow []$
for i in $1 : p$ **do**
  Walks $\leftarrow \{\}$
  $l' \leftarrow l$
  for j in $1{:}|\{\mathcal{G}_1, \mathcal{G}_2, \dots \mathcal{G}_S\}|$ **do**
    if $\left|V_{\mathcal{G}_j}\right| < l'$ **then**
      $l' \leftarrow \left|V_{\mathcal{G}_j}\right|$
    end
    for v in V **do**
      Walks(v) $\leftarrow$ Traceback($\mathcal{G}_j, v, l'$)
    end
  end
  Samples(i) $\leftarrow$ Walks
end
$\mathbf{X} \leftarrow$ Learn Embedding(Samples, $d, \widehat{w}, \alpha, k$)

Algorithm 2: Learning from sequences

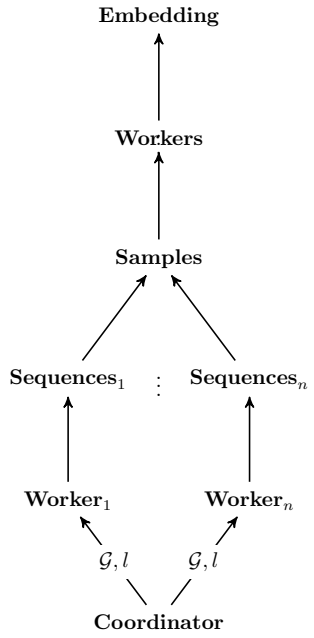

Fig. 4: Outline of parallel processing.

5 Experiments

In our experiments we compare our method D2V with the state of the art N2V [9] method. We look at quality of embeddings and the computational performance. The main observations from the experiments are:

- With increasing size and density of graphs, efficiency of D2V scales better than that of N2V.
- The D2V embedding preserves distances well between most pairs of nodes: in 128 dimensional embedding, over 90% pairs suffer a distortion smaller than 20%. In any dimensions, it performs bettern than N2V.
- Clustering of the D2V embedding works well for community detection, and performs better than N2V measured by the modularity of clusters.
- The representation quality obtained by D2V regarding community detection is quite robust to changes of feature vector dimension, window size, diffusion number, sampled vertex set cardinality and ASGD iteration number.

Computational efficiency. In the first series of experiments we measured the average graph pre-processing and sequence generation times on a Barabasi-Albert graph [2]. Pre-processing in this case involves reading the graph and

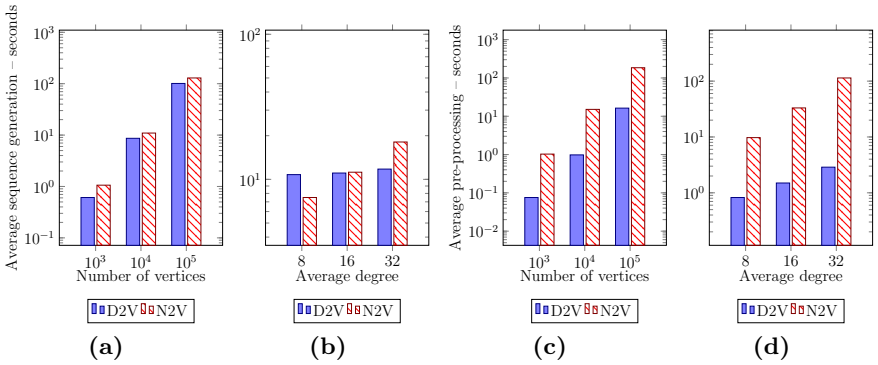

Fig. 5: Barabasi-Albert graph – mean sequence generation and pre-processing time. Columns report mean sequence generation and graph pre-processing time in seconds based on 100 replications. For the graph size benchmarks the number of vertices was set at 10^3, 10^4 and 10^5 while the average degree was fixed to be 10. In case of the average degree experiments the degree was set as 8, 16 and 32 while number of vertices was 10^4. The generated vertex sequences have length 80 and the vertical axis is on a log scale. **(a)** Number of vertices – average sequence generation. **(b)** Average degree – average sequence generation. **(c)** Number of vertices – average pre-processing time. **(d)** Average degree – average pre-processing time. Results imply that our method is robust to graph densification.

creating suitable data structures. N2V in particular requires data structures to regularly update the random walk probabilities. Note that it is the preprocessing and sequence generation where these two methods differ, as they use similar methods for training neural networks.

In separate experiments we modified the size of the graph and the average degree. The results are plotted on Figure 5. Graph the densification slows down N2V considerably both in the sequence generation and pre-processing phase. We could also conclude that on a fixed size dense graph D2V is slightly faster. Similar trends were seen in other models of random graphs.

	Blogcatalog		**PPI**		**Wikipedia**	
	$\|V\|$=10,312		$\|V\|$=3,890		$\|V\|$=4,777	
	$\|E\|$=333,982		$\|E\|$=38,739		$\|E\|$=92,517	
	N2V	D2V	N2V	D2V	N2V	D2V
Sequence generation	59.089	**19.983**	**4.253**	4.684	12.135	**6.879**
Pre-processing	784.899	**3.231**	12.797	**0.362**	185.287	**0.667**

Table 1: Computation time on real life graphs. **BlogCatalog:** Is a social network of bloggers, nodes are bloggers and links are social relationships [1]. **PPI:** is a protein-protein interaction network of humans [5]. **Wikipedia:** Is a word co-occurrence network based on a chunk of the Wikipedia corpus [13]. Columns report running time in seconds extracted from 100 experiments on the datasets. Bold numbers mark the fastest mean pre-processing – sequence generation times on a given dataset.

We tested computation times on a number of real world networks. Again we measured average graph pre-processing and sequence generation times (Table 1). Our results show that on larger networks D2V has a consistent advantage performance wise.

Algorithm	Blogcatalog	PPI	Wikipedia	Flickr	Youtube	Markercafe
Fast Greedy	0.2069	0.3029	**0.1456**	0.4517	–	0.2597
Walktrap	0.1766	0.2571	0.0553	0.4873	–	0.2026
Eigenvector	0.2035	0.2262	0.0915	0.4810	–	0.2455
K-means D2V	**0.2225**	**0.3365**	0.1420	**0.5078**	**0.6265**	**0.2818**
K-means N2V	0.2184	0.3270	0.1376	0.3647	0.4862	0.2630
Hierarchical D2V	0.1610	0.2618	0.0696	–	–	–
Hierarchical N2V	0.1149	0.2774	0.0709	–	–	–

Table 2: Clustering quality measured by modularity. Numbers in the columns represent modularity scores of the clusterings. The baseline community detection algorithms can be found in [6,19,17]. Bold numbers note the highest modularity value obtained on the dataset. Dashes denote missing modularity values when obtaining a clustering was not feasible due to computational complexity of the algorithm. Embeddings were created with baseline parameter settings such that $d = 128,\ n = 10,\ \widehat{w} = 10,\ \alpha = 0.025,\ k = 1,\ l = 40$ (D2V) and $l = 80$ (N2V). The best input and return parameters of N2V were chosen with grid search over $\{0.25, 0.5, 1, 2, 4\}$ while the cluster number varied between 2 and 50. Clusterings with the best modularity were taken. The distance measure was the Euclidean distance in the latent space and hierarchical clustering used Ward's linkage. Besides the earlier used datasets we chose 3 additional social networks to asses the representation quality. **Flickr:** A network of Flickr users [14]. **Youtube:** Is a friendship network of Youtube users [28]. **Markercafe:** Is data from an Israeli social network [7].

Node distance approximation. Using the PPI network we measure how well the shortest path distance of nodes $d(u, v)$ can be approximated by the Euclidean distance of nodes in the embedding space. The relative approximation error $e_{u,v}$ for a given pair of nodes u, v is defined by Equation (4) as the absolute difference between $d(u, v)$ and the scaled Euclidean distance to $d(u, v)$. The factor γ adjusts for the uniform scaling over the graph. We take the γ that minimises the sum of errors.

$$e_{u,v} = \left| \frac{d(u,v) - \gamma \cdot \|\mathbf{X}_v - \mathbf{X}_u\|}{d(u,v)} \right| \tag{4}$$

We plotted cumulative distribution of the relative approximation error for different embedding dimensions on Figure 1a. With a 32 dimensional D2V embedding one can approximate half of the shortest path distances with a relative error below 20%. Increasing the embedding dimension to 128 allows to approximate 90% of shortest paths with an approximation error below 20%. Finally, we also plotted the approximation error obtained with N2V embeddings. A 32 dimensional N2V embedding can only approximate roughly 10% of the shortest path distances with a relative error below 20%. Moreover, increasing the N2V embedding dimension does not decrease the distortion considerably. We conclude that on this graph D2V approximates graph distances better than N2V.

Community detection. We evaluated the utility of the embedding in community detection. We clustered the embedded nodes in the embedding space using

k-means clustering, and then computed the modularity [17] of the clusters as a quality measure. The experiments involved six different datasets with number of vertices ranging from few thousands to millions and we compared our results to clusterings obtained with standard community detection methods.

Results are seen in Table 2. We used k-means clustering and hierarchical clustering to extract node groups. Our results show that k-means clustering of the embeddings outperforms all other methods on most of the datasets. Moreover, D2V (our method) results in clusterings that are higher quality than clusters created with N2V.

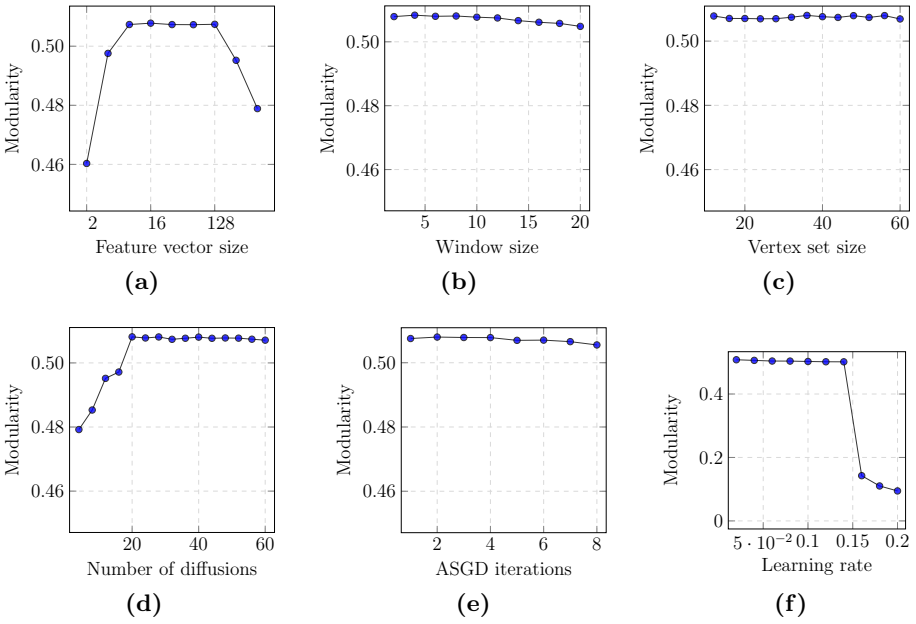

Fig. 6: Sensitivity of k-means clustering performance to change of embedding method parameters. Embeddings were created with baseline parameter settings such that $d = 128$, $n = 10$, $\widehat{w} = 10$, $\alpha = 0.025$, $k = 1$ and $l = 40$. Manipulated parameters are on the horizontal axis and clustering performance measured by modularity is on the verticals. **(a)** Feature vector size. **(b)** Window size. **(c)** Sampled vertex set size. **(d)** Number of diffusions. **(e)** ASGD iterations. **(f)** Initial learning rate.

To test the robustness of the representation quality we carried out a complete sensitivity analysis on the Flickr dataset. We manipulated parameters of the embedding procedure and we observed how the modularity of k-means clustering of the nodes changes. Our results are on subplots of Figure 6. The experimental results support that the representation quality is robust to feature vector size, window size, diffusion number, sampled vertex set cardinality and ASGD iteration number changes. We also observe a sudden drop in cluster quality as the learning rate is increased above 0.14.

6 Conclusions

In this work we proposed *Diff2Vec* a node sequence based graph embedding model that uses diffusion processes on graphs to create vertex sequences. We implemented this method in Python and demonstrated that the design of the algorithm results in fast sequence creation in realistic settings. It also allows parallel vertex sequence generation which leads to additional speed up. We supported evidence that the computational performance of our method is robust to graph densification and growth. We confirmed that node features created with *Diff2Vec* are useful features for downstream machine learning tasks. We gave a detailed evaluation of the representation quality of embeddings on shortest path distance approximation and the machine learning task of community detection. Our findings show that besides the favourable computational performance the representation quality itself is competitive with other methods. We conclude that our work is an important contribution towards solving large scale network analysis problems.

References

1. Nitin Agarwal, Huan Liu, Sudheendra Murthy, Arunabha Sen, and Xufei Wang. A social identity approach to identify familiar strangers in a social network. In *ICWSM*, 2009.
2. Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
3. Noga Alon, Chen Avin, Michal Koucký, Gady Kozma, Zvi Lotker, and Mark R Tuttle. Many random walks are faster than one. *Combinatorics, Probability and Computing*, 20(4):481–502, 2011.
4. Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.
5. Andrew Chatr-Aryamontri, Bobby-Joe Breitkreutz, Rose Oughtred, Boucher, et al. The biogrid interaction database: 2015 update. *Nucleic acids research*, 43(D1):D470–D478, 2014.
6. Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.
7. Michael Fire, Lena Tenenboim, Ofrit Lesser, Rami Puzis, Lior Rokach, and Yuval Elovici. Link prediction in social networks using computationally efficient topological features. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 73–80. IEEE, 2011.
8. Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *arXiv preprint arXiv:1705.02801*, 2017.
9. Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.
10. Ivan Herman, Guy Melançon, and M Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43, 2000.

11. Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
12. Jon Kleinberg, Aleksandrs Slivkins, and Tom Wexler. Triangulation and embedding using small sets of beacons. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 444–453. IEEE, 2004.
13. Matt Mahoney. Large text compression benchmark, 2011.
14. Julian McAuley and Jure Leskovec. Image labeling on a network: Using social-network metadata for image classification. In *Computer Vision-ECCV*, pages 828–841, 2012.
15. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
16. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
17. Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.
18. Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
19. Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218, 2006.
20. Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.
21. Rik Sarkar, Xiaotian Yin, Jie Gao, Feng Luo, and Xianfeng David Gu. Greedy routing with guaranteed delivery using ricci flows. In *Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on*, pages 121–132. IEEE, 2009.
22. Yi Shang, Wheeler Ruml, Ying Zhang, and Markus PJ Fromherz. Localization from mere connectivity. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 201–212. ACM, 2003.
23. David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.
24. Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.
25. Duncan J Watts and Steven H Strogatz. Collective dynamics of'small-world'networks. *nature*, 393(6684):440, 1998.
26. Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.
27. Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graphs. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 274–285. SIAM, 2005.
28. Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.