SINr: Fast Computing of Sparse Interpretable Node Representations is not a Sin!

Thibault Prouteau, Victor Connes, Nicolas Dugué, Anthony Perez, Jean-Charles Lamirel, Nathalie Camelin, Sylvain Meignier

SINr: fast computing of Sparse Interpretable Node Representations is not a sin!

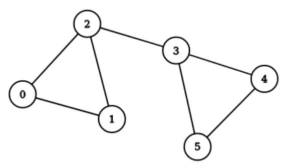
Thibault Prouteau[1], Victor Connes[2], Nicolas Dugué[1], Anthony Perez[3], Jean-Charles Lamirel[4], Nathalie Camelin[1], and Sylvain Meignier[1]

[1] Le Mans Université, LIUM, EA 4023, Laboratoire d'Informatique de l'Université du Mans, France `nicolas.dugue@univ-lemans.fr`
[2] Laboratoire des Sciences du Numérique de Nantes, Université de Nantes, France `victor.connes@univ-nantes.fr`
[3] Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, FR-45067 Orléans, France `anthony.perez@univ-orleans.fr`
[4] Université de Strasbourg, LORIA, Equipe Synalp, France `lamirel@loria.fr`

Abstract. While graph embedding aims at learning low-dimensional representations of nodes encompassing the graph topology, word embedding focus on learning word vectors that encode semantic properties of the vocabulary. The first finds applications on tasks such as link prediction and node classification while the latter is systematically considered in natural language processing. Most of the time, graph and word embeddings are considered on their own as distinct tasks. However, word co-occurrence matrices, widely used to extract word embeddings, can be seen as graphs. Furthermore, most network embedding techniques rely either on a word embedding methodology (`Word2vec`) or on matrix factorization, also widely used for word embedding. These methods are usually computationally expensive, parameter dependant and the dimensions of the embedding space are not interpretable. To circumvent these issues, we introduce the Lower Dimension Bipartite Graphs Framework (`LDBGF`) which takes advantage of the fact that all graphs can be described as bipartite graphs, even in the case of textual data. This underlying bipartite structure may be explicit, like in coauthor networks. However, with `LDBGF`, we focus on uncovering *latent* bipartite structures, lying for instance in social or word co-occurrence networks, and especially such structures providing conciser and interpretable representations of the graph at hand. We further propose `SINr`, an efficient implementation of the `LDBGF` approach that extracts Sparse Interpretable Node Representations using community structure to approximate the underlying bipartite structure. In the case of graph embedding, our near-linear time method is the fastest of our benchmark, parameter-free and provides state-of-the-art results on the classical link prediction task. We also show that low-dimensional vectors can be derived from `SINr` using singular value decomposition. In the case of word embedding, our approach proves to be very efficient considering the classical similarity evaluation.

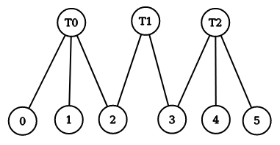
Introduction

The aim of *graph embedding* is to learn representations of nodes encompassing the structural properties of the network. Such representations (or *embedding vectors*) can then be processed in reduced time and space and proved to be useful to deal with problems such as link prediction [14] and node classification [2]. Among the state-of-the art embedding methods, techniques adapting the Word2vec (for semantic representations of words) framework [15] with random walks have been extensively studied [19, 20]. Matrix factorization approaches [1, 6, 17] were also widely considered, some of them also inspired by the GloVe word embedding technique [5]. As one can see, natural language processing (NLP) literature influenced a lot the field of graph embedding, providing a solid base on which to build. Indeed, word embedding also aims to provide dense continuous representations, those being used as inputs of NLP systems for opinion mining, translation or text categorization for instance.



$$\begin{pmatrix} 0&1&1&0&0&0 \\ 1&0&1&0&0&0 \\ 1&1&0&1&0&0 \\ 0&0&1&0&1&1 \\ 0&0&0&1&0&1 \\ 0&0&0&1&1&0 \end{pmatrix}$$

(a)



$$\begin{pmatrix} 1&0&0 \\ 1&0&0 \\ 1&1&0 \\ 0&1&1 \\ 0&0&1 \\ 0&0&1 \end{pmatrix}$$

(b)

Fig. 1. Illustrating the LDBGF: (a) a toy graph and its adjacency matrix (b) a low-dimensional bipartite graph representation. The adjacency matrix of the $\perp$ nodes (rows) from the toy graphs are represented with their links to the $\top$ nodes (columns). The resulting adjacency matrix is smaller and is actually an interpretable embedding: dimensions represent the connectivity to $\top$ nodes which are tangible entities.

Our approach. To the best of our knowledge, despite the connections between these tasks, graph and word embedding are usually considered as distinct tasks. In this paper, we tackle both of these tasks, considering that word co-occurrence matrices extracted from substantial text corpora can also be seen as graphs. We thus reduce the task of learning word embedding to a graph embedding problem and provide a unified approach for both tasks. Furthermore, in all aforementioned graph embedding works, authors mainly focus on getting efficient and low-dimensional dense vectors, somehow neglecting the time complexity and the interpretability of the results. Meanwhile, interpretability and green computing are major issues for the machine learning field. Taking into account these considerations, we introduce the Lower Dimension Bipartite Graphs Framework (LDBGF), a framework based on low-dimensional bipartite representations of graphs to compute sparse interpretable vectors for words and nodes.

Guillaume and Latapy [8] showed that all complex networks can be represented by a bipartite structure. Coauthoring networks are by nature represented as a bipartite graph $G = (\top, \bot, E)$ where $\top$ corresponds to the set of papers and $\bot$ to the set of authors, each author being connected to papers they co-signed. The unipartite coauthoring network can thus be retrieved by *projecting* the bipartite graph, *i.e.* by adding an edge between any two authors linked to the same paper. When considering social networks, one can reasonably assume that there is a latent bipartite structure [8]: people are connected through their school, family, firm, *etc.* In the case of word co-occurrence networks extracted from large corpora, words are connected by syntactic rules, but also thematic or semantic fields. With `LDBGF`, we assume that we can uncover low-dimensional latent bipartite structures for real networks, representing them in a concise way (Fig. 1). In such a case, the graph embedding space is interpretable: nodes are represented by their connectivity to the $\top$-nodes uncovered that are tangible graph objects (Fig. 1(b)). Guillaume and Latapy [8] use cliques in the $\top$-part of the bipartite graph to encode relations between nodes. The problem considered can thus be seen as a CLIQUE COVERING problem [8]: given a network, compute a set of cliques such that each edge belongs to at least one clique (ensuring that the $\bot$-projection yields the original network). However, `LDBGF` enforces the number of cliques to be as small as possible in order to represent the original graph with as few $\top$-nodes as possible, and thus get a lower-dimensional embedding space. For instance, considering each edge as a clique would yield too many cliques. Thus, `LDBGF` is related to MINIMUM CLIQUE COVERING NUMBER, a classical NP-Hard problem [16]. Hence, one needs heuristics to obtain a satisfying covering in a reasonable time.

Our contribution. In line with the `LDBGF` philosophy, we introduce the `SINr`[5] (SPARSE INTERPRETABLE NODE REPRESENTATIONS) method that derives sparse and interpretable word and graph embedding vectors in near-linear time, thus making a parsimonious usage of CPU resources. The main idea of `SINr` is to use the community structure to provide an approximate solution to the MINIMUM CLIQUE COVERING problem. Indeed, communities can be seen as clique relaxations that can be uncovered very efficiently [3], allowing `SINr` to run faster than the other graph embedding approaches. Basing our work on the hypothesis that nodes with similar connectivity to the communities are similar, we compute our bipartite approximation directly from community structure. This is why `SINr` vectors are sparse and interpretable: each dimension of the embedding vector represents the node connectivity to a community, and the number of dimensions is thus linear in the number of communities. Furthermore, if low-dimensional vectors are required, our `SINr` vectors can be projected in a lower-dimensional space using standard dimension reduction techniques whilst preserving their efficiency.

Outline. We first describe our `SINr` embedding technique (Section 1), and how it can be applied to deal with word and graph embeddings. We then detail the

[5] `https://www.github.com/anthonimes/SINr`

experimental setup (Section 2), considering graph and textual datasets, and detailing graph and word embedding approaches we compare to. We finally present the very encouraging experimental results on the classical link prediction task for graph embedding, and on the similarity task for word embedding (Section 3). We also demonstrate the lower computational cost of `SINr` when compared to the other graph embedding methods we consider.

1 `SINr`: Algorithmic framework

`SINr` is a near-linear time implementation of `LDBGF`. Since communities can be seen as cliques relaxations [3], instead of relying on cliques to compute a latent bipartite structure (Fig 2(b)), `SINr` relies on communities which are faster to compute. However, unlike in cliques, nodes connected to a community may not be connected to the whole community. To cope with this problem, we use a recent work by a subset of the same authors, allowing to account for the nodes connectivity to the community structure, the Node F-Measure framework [7]. The aim of our method is to produce embedding vectors where nodes with a similar behaviour towards the community structure of the network lie close in the embedding space. Thus, the approach of `SINr` actually consists in two steps: the first one is detecting the p communities of the network at hand (Section 1.1). The second, described Section 1.2, is computing the Node Predominance and Node Recall of each node for every community (Fig 2(c)). This allows to obtain a $2p$ embedding vector for each node (Fig 2(d)). This explains why `SINr` leads to sparse and interpretable vectors: dimensions of the embedding space are not abstract but community-related. In the remaining of this section, we describe further the two steps of `SINr`.

1.1 First step: community detection algorithms

We consider (un)directed and (un)weighted graphs $G = (V, E, W)$ with V the set of vertices, E the set of edges and W the weights attached to the edges. The neighbourhood of a vertex $v \in V$ is denoted by $N(v)$, and we let $d(v) = \sum_{u \in N(v)} W_{u,v}$. The community structure of G is a *partition* $\mathcal{C} = \{C_1, \ldots, C_p\}$ of V such that each subgraph $G[C_i]$ is densely connected, while the density between C_i and C_j is low, $1 \leqslant i < j \leqslant p$. To measure the quality of such a partition, Girvan and Newman [3] introduced the modularity $Q_{\mathcal{C}}$:

$$Q_{\mathcal{C}} = \frac{1}{2m} \cdot \sum_{i,j} \left[A_{ij} - \frac{d_i \cdot d_j}{2m} \right] \cdot \delta(C_i, C_j) \tag{1}$$

where A_{ij} denotes the existence (or weight) of edge ij, and the δ-function $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise. Computing a partition which maximizes modularity is NP-Hard [4] but there exist community detection algorithms which implement heuristics to maximize modularity, such as the extensively used Louvain algorithm [3].

Community structure was already successfully considered to extract graph embedding vectors: Bhowmick et al. [2] used a projection of the hierarchical community tree to obtain the network embeddings. However, our SINr approach relies directly on the community structure, defining the embedding space using the connectivity to the communities. We considered using various algorithms to detect the communities and thus ran several tests using Label Propagation [21] (LP), Infomap, and Louvain algorithms. Both LP and Louvain run fast (near-linear time), while Infomap is slower but known to perform better. Our experiments have shown that the graph embeddings extracted using Louvain with the SINr methodology perform better for link prediction while the word embeddings extracted with LP perform better on words similarity (Section 3).

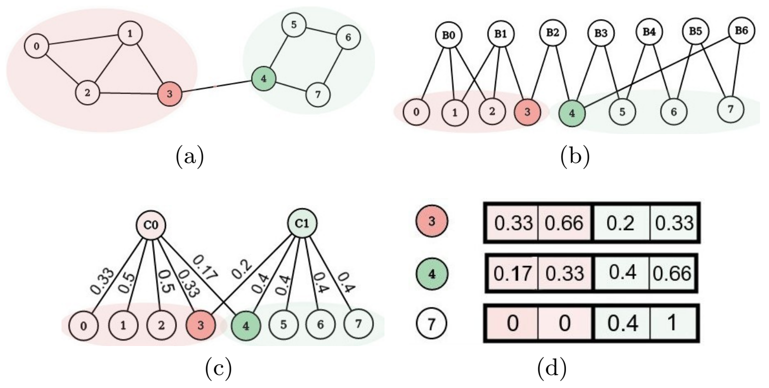


Fig. 2. SINr : (a) Toy example with oval shapes representing communities (b) Latent bipartite structure using maximum cliques (c) Approximation of such a structure using detected communities and Node predominance (d) SINr vectors with Node predominance and Node recall for both communities.

1.2 Second step: Node Predominance and Node Recall

The Node F-measure framework introduced in [7] encompasses interesting topological properties such as centrality and community roles. It is based on **Node Predominance** (NP) and **Node Recall** (NR). Node Predominance (Eq. 2) is used to characterize the node's connectivity towards its community: the higher it is, the more a node is connected to its community. Node Recall (Eq. 3) is used to evaluate the connectivity of a node with nodes outside its community: the weaker it is, the more the node is connected to the outside. Considering $\mathcal{C} = \{C_1, \ldots, C_p\}$, and $1 \leqslant i \leqslant p$, we define NP and NR for each node u as:

$$\mathrm{NP}_i(u) = \frac{d_{C_i}(u)}{d_{C_i}} \quad (2) \qquad \mathrm{NR}_i(u) = \frac{d_{C_i}(u)}{d(u)} \quad (3)$$

where $d_{C_i}(u)$ is the degree of node u in C_i, *i.e.* $d_{C_i}(u) = |\{uv \in E \mid v \in C_i\}|$, and d_{C_i} is the number of edges incident to vertices in community C_i.

In this work, we consider NP and NR for a given node to *any* community of a partition. We thus compute $2p$-dimensional embeddings (Fig 2(d)) that encompass connectivity of the nodes towards the network's communities.

2 Experimental setup

2.1 Datasets

Networks considered for graph embedding. To conduct our experiments we use several well-known datasets from different fields described by their size ($n = |V|$ and $m = |E|$). For the sake of our experiments, we extract the largest connected component of each graph and consider them undirected.

(a) `Citeseer` [13] (`Cts`, $n = 3,312$ and $m = 4,660$) and `Cora` [13] ($n = 2,708$ and $m = 5,278$) are networks of citations of scientific publications.
(b) `Email-eu` [11] (`Eu`, $n = 1,005$ and $m = 16,706$) is a network representing a sender-receiver relationship w.r.t. e-mails within European research institution.
(c) `arXiv` [9] ($n = 18,771$ and $m = 198,050$) covers scientific collaborations between authors of papers submitted in the Astrophysics category.
(d) `Facebook` (`Fb`, $n = 63,731$ and $m = 817,035$) represents friendship data of Facebook users.

Corpora considered for word embedding. Our experiments on word embedding span two corpora. For each corpus[6], words with fewer than four occurrences and stop words (words without semantic interest, *e.g.* the, at, which) are removed. Furthermore, both corpora are lemmatized using `SpaCy`.

(a) `text8` is made of the first 100MB of Wikipedia's March 2006 dump. After preprocessing, the corpus contains $6,039,538$ tokens for a vocabulary of $73,860$ words.
(b) `OANC`, the *Open American National Corpus* is a 15-million-word corpus of written and spoken American English. After preprocessing, the number of tokens of this corpus is $6,016,207$ for a vocabulary of $53,282$ words.

2.2 Extracting word co-occurrence networks for textual corpora

When dealing with textual data, the first step is to extract a word co-occurrence network from the corpora at hand. To build this graph, in a similar way to other textual word embedding methods [12, 18], we first compute the word co-occurrence matrix by applying a sliding context window to each sentence in the corpora. At the second step, in order to filter out the insignificant co-occurrences from this matrix, we compute the *Pointwise Mutual Information* (PMI [12], Eq. 4) for each entry of the matrix, and set entries to 0 when the PMI value is negative. When PMI value is positive, the entry remains unchanged.

[6] http://mattmahoney.net/dc/textdata.html and http://www.anc.org/data/oanc/

Let w_1, w_2 be two words, $p(w_1)$ the frequency of w_1 in the corpora and $p(w_1, w_2)$ the co-occurrence frequency of both w_1, w_2. The PMI is defined as follows:

$$\text{PMI}(w_1, w_2) = \log_2 \left(\frac{p(w_1, w_2)}{p(w_1)p(w_2)} \right) \tag{4}$$

The third step consists in building a weighted graph $G = (V, E, W)$ from the *filtered* matrix. The vertex set V represents words. The edge set E and the weights W attached to the edges represent the co-occurrences in the corpora: there is an edge between two vertices if these words co-occur significantly in the corpus, and the weight attached to the edge is the number of these co-occurrences.

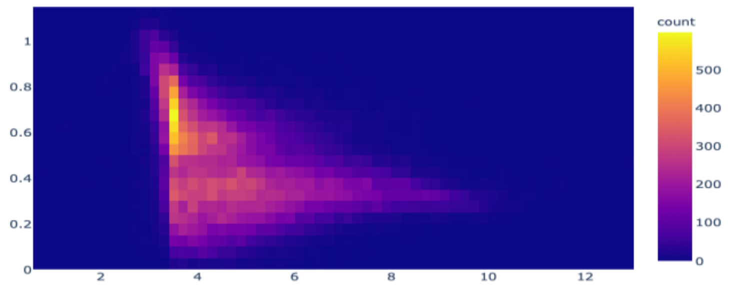


Fig. 3. Heatmap of the weighted degrees of the graph extracted on `OANC` after IPMI according to the weighted degrees before applying IPMI, abscissa in logarithmic scale.

The final step consists in applying a re-weighting scheme to the graph. According to our extensive experiments, such a process is necessary to extract relevant communities and thus word embeddings. We thus introduce our original `IPMI` (Iterative Pointwise Mutual Information) re-weighting scheme. Let E_{ord} be the edges (u, v) of E ordered from the highest to the lowest sum of its weighted degrees $d(u)+d(v)$. For each edge (u, v) ordered as in E_{ord}, we iteratively update $W_{u,v}$ with the `IPMI` value:

$$\text{IPMI(u,v)} = \frac{W_{u,v}}{d(u)d(v)} \tag{5}$$

To illustrate the effect of the `IPMI`, we consider the graph extracted from the `text8` corpus described Section 2. We plot the weighted degrees of the vertices of this graph after `IPMI` against the ones before `IPMI`. As one can see Figure 3, the plot actually looks like an *inverse* function. Our experiments thus seem to show that it is important to considerably lower the weights of the hubs (nodes that are widely connected). Our hypothesis is that hubs represent very frequent words, which are more susceptible to be polysemous. It may thus be more relevant to detect communities based on more specific words by lowering the hubs influence before the community detection.

2.3 State-of-the-art algorithms

Graph embedding. We use several state-of-the-art graph embedding techniques provided by the `karateclub` library [23] to compare with the results obtained by our method. We use the default implementation parameters and embedding vectors dimension number is always set to 128.

(a) `HOPE` [17] stands for *High-order proximity preserved embedding.* This approach uses generalized SVD to approximate proximity matrices such as the ones that can be obtained by using Katz or Adamic-Adar indexes. In our experiments, the proximity matrix used is the Common neighbours one.
(b) `Deepwalk` (`DW`) [19] is similar in spirit to the `Word2vec` Skip-gram approach [15]. Instead of computing Skip-gram on sentences, the authors compute it on paths given by random walks on the graph. The number of walks is fixed to 10, the walk length to 80 and the window size to 5.
(c) `Walklets` (`WL`) [20] is also based on random walks, but the authors introduce a sampling method improving the results. The parameters are the same as for `Deepwalk` except that the window size is 4.

We also ran experiments with `Diff2vec` [23], `GraRep` [6] and `LaplacianEigenMaps` [1] but for the sake of concision, we do not report their results. Indeed, `GraRep` is similar to `Walklets` in spirit, and obtain similar results but it runs slower and requires much more memory. `LaplacianEigenmaps` and `Diff2vec` obtain poorer results than the other approaches.

Word embedding. We compare to classical state-of-the-art methods but do not consider contextual word embeddings, since our work focuses on interpretability and parsimonious usage of computing resources. The implementations are detailed below. In all of them, the sliding context window is set to 5 words and the number of dimensions to 300.

(a) `Word2vec` (`W2V`) is one of the most popular methods to learn word embeddings [15]. We compute embeddings for each corpus using the `CBOW` architecture provided by `Gensim` [22] .
(b) `GloVe` [18] stands for *Global Vectors for Word Representation* and aims at providing corpus-derived semantic models based on global word co-occurrence statistics. We use the `GloVe` implementation provided by the authors.
(c) `SVD2vec` is an implementation [7] of the approach described by Levy et al. [12] using *SVD* on a *PPMI* co-occurrence matrix to compute embeddings.

3 Experiments and results

3.1 `SINr` is fast: Complexity and runtime

Given the community structure of a network, computing embeddings can be done in linear time since one only needs to parse the edges of the graphs to compute

[7] https://git-lium.univ-lemans.fr/vpelloin/svd2vec

Node Predominance and Node Recall. As stated previously, we use Louvain's algorithm to compute graph embeddings, which is known to run in $O(m)$ time [2, 24]. For word embedding, we use Label propagation which is known to run in near-linear time, each propagation step running in $O(m)$ time and the algorithm converging in few steps. Altogether, the running time of our method is thus in $O(m)$. To conclude this section, we present the average time[8] (50 runs) needed to compute embeddings for `SINr` and graph embedding algorithms `HOPE`, `Deepwalk` and `Walklets` we compare to (see Section 2.3 for more details). As detailed Section 2.1, the `Facebook` graph contains $63,731$ vertices and $817,035$ edges. As one can see, our method is significantly faster than the other approaches.

Table 1. Average runtime in seconds (left) and CPU time taking into account parallelism (right)

	`Cora`	`Eu`	`Cts`	`arXiv`	`Fb`
`SINr`	**0.3/1.3**	**0.3/2**	**0.1/0.9**	**0.9/4**	**3/8**
`HOPE`	0.2/3	0.6/8	0.7/2	10/120	26/195
`DW`	24/36	13/18	20/30	264/378	336/422
`WL`	26/38	12/18	24/36	261/365	475/652

3.2 Graph embedding: Link Prediction

Problem description. Let $G = (V, E)$ be a simple undirected network. Let U denote the universal set containing $\frac{n(n-1)}{2}$ possible pairs of V and $\overline{E} = U \setminus E$ the set of non-edges of G. We consider link prediction as a binary classification problem where we assume that there are some missing links (or links that will appear later on) and we train a classifier to detect such links with high probability.

As in [17], we randomly separate the graph into a *training set* (containing 80% edges) and a *test set* containing the remaining edges. We train the embedding vectors on the training set, and then evaluate link prediction on the test set. Training is done using `Xgboost` and `Logistic regression` and the best results are kept in each case. For the sake of comparison, we consider the node representation algorithms detailed Section 2.3 with Hadamard product, as well as the following set of heuristic features which is proved to be efficient on the link prediction task (see [14]): Common Neighbours, Adamic Adar, Preferential Attachment, Jaccard index, Resource Allocation Index. The vector obtained from such features is referred to as the *heuristics* vector.

Evaluation. We consider two evaluation procedures. In the first one, the test set is made of the 20% of existing edges augmented with the same number of negative examples sampled from $\overline{E}$. We run each test 50 times and present the

[8] With two Intel Xeon CPU E5-2660 2.20GHz : 16 cores, 96Go Ram.

averaged accuracy. `SINr` achieves the highest accuracy for most of the small datasets as one can see Table 2. On sizeable graphs, results are comparable to those of the other state-of-the-art approaches. On such graphs, the communities may be harder to detect, explaining the results. Furthermore, we can observe that results are still close to the best ones, even when applying SVD on `SINr` in order to get 20 dimensions. Our approach can thus provide both sparse interpretable and dense low-dimensional vectors.

	`SINr`	`SINr-SVD`	Heuristics	`DW`	`WL`	`HOPE`
`Cora`	**0.83**	**0.83**	0.76	0.73	0.82	0.75
`Eu`	**0.88**	**0.88**	0.87	0.81	0.87	0.87
`Cts`	**0.88**	0.86	0.78	0.76	0.87	0.83
`arXiv`	0.92	0.90	**0.97**	0.92	0.96	0.92
`Fb`	0.91	0.89	**0.93**	0.86	0.92	0.90

Table 2. Accuracy on the link prediction task. `SINr-SVD` is 20-d `SINr` vectors obtained using SVD.

	p	$Q_{\mathcal{C}}$	σ
`Cora`	23	0.80	0.94
`Eu`	6	0.41	0.44
`Cts`	33	0.85	0.96
`arXiv`	28	0.62	0.87
`Fb`	73	0.62	0.96

Table 3. Number of communities p, modularity $Q_{\mathcal{C}}$, and sparsity coefficient σ of the `SINr` vectors.

In our second evaluation procedure, we consider a test set with many more negative examples. Due to the high cardinality of the set $\overline{E}$, we randomly sample about 1% of such pairs for evaluation on most graphs, except for `Facebook` where we sample 0.1%. We run each test 50 times and present the averaged `precision@`k. The latter measures the fraction of node pairs in the top-k most probable pairs (according to our classifier) corresponding to an actual edge in the network. Figure 4 presents representative results. For most datasets introduced Section 2.1, experiments highlight the relevance of our method that competes with the state-of-the art approaches. On `Email-eu`, the lower modularity of the partitions returned (Table 3) degrades the performances of our approach. Our method obtains these encouraging results being by far the fastest (Table 1).

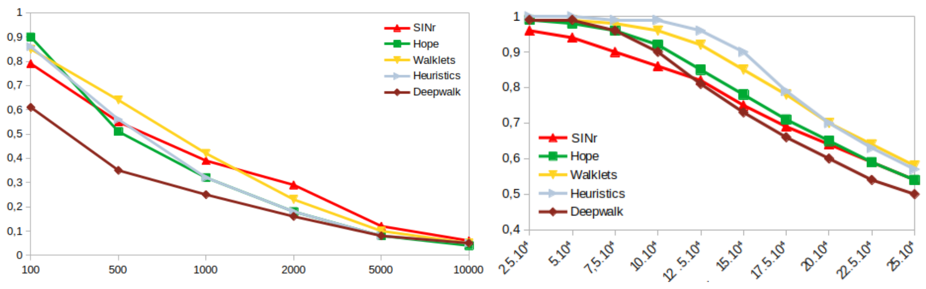


Fig. 4. `precision@`k, the proportion among the k most probable pairs that are edges according to the classifier trained on link prediction on `Cts` (at left) and `Fb` (at right).

3.3 Word embedding: Similarity

To evaluate the performances of SINr word embeddings, we consider a word similarity task with datasets presented in Lastra-Diaz et al. [10]. Each dataset is composed of pairs of words associated with a similarity value assessed by humans. The evaluation consists in computing the Spearman correlation between the cosine similarity for all the pairs in the embedding space and the human value.

text8	W2V	GloVe	SVD2vec	SINr
MC28	0.58	0.42	0.67	**0.69**
RG65	0.52	0.48	0.57	**0.64**
MTurk771	**0.52**	0.48	0.45	0.48
WS353Rel	0.47	0.44	0.46	**0.50**
WS353Full	**0.55**	0.47	**0.55**	0.53
MEN	0.53	0.48	**0.61**	0.52

OANC	W2V	GloVe	SVD2vec	SINr
MC28	0.45	0.54	0.33	**0.62**
RG65	0.33	0.32	0.32	**0.39**
MTurk771	**0.44**	0.39	0.36	0.37
WS353Rel	0.40	0.34	**0.47**	0.41
WS353Full	0.49	0.40	**0.51**	0.44
MEN	0.44	0.46	**0.59**	0.40

Table 4. Word similarity evaluation in Spearman correlation between human judgement and cosine similarity for each model. At left, on text8 corpus, at right on OANC.

The first four datasets (MC28, RG65, MTurk771and WS353Rel) are composed of pairs of names, the last two datasets (WS353Full, MEN) contain names, adjectives and verbs. On both text8 and OANC, SINr word embeddings perform best on MC28 and RG65 whose words in pairs have a similar meaning. W2V and SVD2vec achieve better results than SINr on datasets with related pairs of words (MTurk771 and WS353Rel) and datasets with names adjectives and verbs (WS353Full, MEN). Overall, the performances of SINr on this task remain encouraging and show that word embeddings extracted from co-occurrence networks can achieve good results on the similarity task.

4 Conclusion

We introduced LDBGF, a novel approach based on the underlying bipartite structure of networks to compute sparse interpretable embeddings. Moreover, we developed SINr, a near linear-time implementation of this newly introduced framework that is faster than all the other graph embedding approaches we compare to. Although embedding techniques explicitly aim to get dense low-dimensional vectors, we demonstrate that the SINr vectors achieve state-of-the-art results on classic graph and word embedding evaluation tasks whilst maintaining a sparse and interpretable representation. We also show that projecting our embedding vectors into a low-dimensional space using SVD does not significantly lower the results. These results demonstrate the relevance of the LDBGF philosophy. We thus hope for new efficient implementations of this framework. For instance, it would be interesting to extend our approach to deal with temporal networks by considering incremental clustering algorithms.

References

1. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: NIPS. pp. 585–591 (2002)
2. Bhowmick, A.K., Meneni, K., Danisch, M., Guillaume, J., Mitra, B.: Louvainne: Hierarchical louvain method for high quality and scalable network embedding. In: WSDM. pp. 43–51 (2020)
3. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech.: Theory Exp. **2008**(10), P10008
4. Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: Maximizing modularity is hard. arXiv preprint physics/0608255 (2006)
5. Brochier, R., Guille, A., Velcin, J.: Global vectors for node representations. In: WWW. pp. 2587–2593 (2019)
6. Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: CIKM. pp. 891–900 (2015)
7. Dugué, N., Lamirel, J.C., Perez, A.: Bringing a feature selection metric from machine learning to complex networks. In: Complex networks (2). pp. 107–118 (2018)
8. Guillaume, J.L., Latapy, M.: Bipartite graphs as models of complex networks. Physica A **371**(2), 795–813 (2006)
9. Kunegis, J.: The koblenz network collection. In: WWW. pp. 1343–1350 (2013)
10. Lastra-Díaz, J.J., Goikoetxea, J., Hadj Taieb, M.A., García-Serrano, A., Aouicha, M.B., Agirre, E.: Reproducibility dataset for a large experimental survey on word embeddings and ontology-based methods for word similarity. Data in Brief **26**, 104432 (2019)
11. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graph evolution: Densification and shrinking diameters. ACM Trans. Kno. Discov. from Data **1**(1), 2–es (2007)
12. Levy, O., Goldberg, Y., Dagan, I.: Improving distributional similarity with lessons learned from word embeddings. ACL **3**, 211–225 (2015)
13. Lu, Q., Getoor, L.: Link-based classification. In: ICML. pp. 496–503 (2003)
14. Martínez, V., Berzal, F., Talavera, J.C.C.: A survey of link prediction in complex networks. ACM Comput. Surv **49**(4), 69:1–69:33 (2017)
15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS. pp. 3111–3119 (2013)
16. Monson, S.D., Pullman, N.J., Rees, R.: A survey of clique and biclique coverings and factorizations of (0, 1)-matrices. Bull. Inst. Combin. Appl **14**, 17–86 (1995)
17. Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W.: Asymmetric transitivity preserving graph embedding. In: SIGKDD. pp. 1105–1114 (2016)
18. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: EMNLP. pp. 1532–1543 (2014)
19. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: SIGKDD. pp. 701–710 (2014)
20. Perozzi, B., Kulkarni, V., Chen, H., Skiena, S.: Don't walk, skip! online learning of multi-scale network embeddings. In: ASONAM. pp. 258–265 (2017)
21. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Phys review E **76**(3), 036106 (2007)
22. Řehůřek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In: LREC. pp. 45–50 (2010)
23. Rozemberczki, B., Kiss, O., Sarkar, R.: An api oriented open-source python framework for unsupervised learning on graphs (2020)
24. Traag, V.A.: Faster unfolding of communities: Speeding up the louvain algorithm. Physical Review E **92**(3), 032801 (2015)