

An Effective and Efficient Graph Representation Learning Approach for Big Graphs

Edoardo Serra¹, Mikel Joaristi², Alfredo Cuzzocrea³, Selim Soufargi⁴ and Carson K. Leung⁵

¹Computer Science Dept., Boise State University, Boise, ID, USA

²Computer Science Dept., Boise State University, Boise, ID, USA

³iDEA Lab, University of Calabria, Rende, Italy

⁴iDEA Lab, University of Calabria, Rende, Italy

⁵Computer Science Dept., University of Winnipeg, MB, Canada

Abstract

In the Big Data era, large graph datasets are becoming increasingly popular due to their capability to integrate and interconnect large sources of data in many fields, e.g., social media, biology, communication networks, etc. Graph representation learning is a flexible tool that automatically extracts features from a graph node. These features can be directly used for machine learning tasks. Graph representation learning approaches producing features preserving the structural information of the graphs are still an open problem, especially in the context of large-scale graphs. In this paper, we propose a new fast and scalable structural representation learning approach called SparseStruct. Our approach uses a sparse internal representation for each node, and we formally proved its ability to preserve structural information. Thanks to a light-weight algorithm where each iteration costs only linear time in the number of the edges, SparseStruct is able to easily process large graphs. In addition, it provides improvements in comparison with state of the art in terms of prediction and classification accuracy by also providing strong robustness to noise data.

Keywords

Graph Representation Learning, Sparse Representation, Large-Scale Graph Structural Representation

1. Introduction

Many fields are naturally connected with graph structures, including social networks, biological interactions, and communication networks. With the advent of Big Data, there is a huge availability of large-scale graph datasets, on top of with making analytics (e.g., [1]), also considering privacy/security issues (e.g., [2]). Given this availability, there is a great need to apply machine learning techniques on large scale graphs.


Graph node representation learning is a powerful tool to automatically extract features from the nodes inside a graph. Such features are the input of standard machine learning tasks that can solve several node classification tasks (e.g., in [3] and [4] authors use graphs describing relationships between entities to identify malicious/bad actors inside such graphs).

SEBD 2021: The 29th Italian Symposium on Advanced Database Systems, September 5-9, 2021, Pizzo Calabro (VV), Italy

✉ edoardoserra@boisestate.edu (E. Serra); mikeljoaristi@boisestate.edu (M. Joaristi); alfredo.cuzzocrea@unical.it (A. Cuzzocrea); selim.soufargi@unical.it (S. Soufargi); kleung@cs.umanitoba.ca (C. K. Leung)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

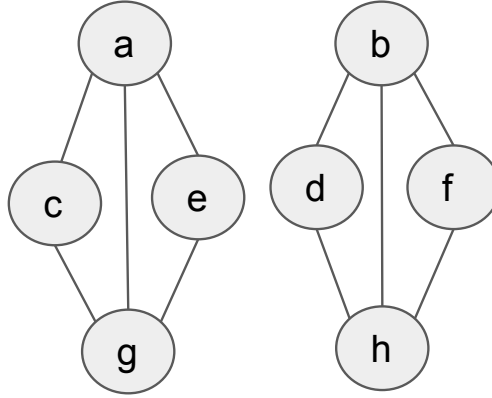


Figure 1: Structural vs. connectivity proximity.

More specifically, a graph node representation associates to each graph node a vector of hidden features. We consider only representation learning models trained with an unsupervised procedure that is independent to any specific machine learning task.

Unsupervised graph representation learning approaches are divided into connectivity and structure-based. Connectivity-based representation learning methods work on the assumption that nodes that are in proximity have similar representation, e.g., nodes *a* and *c* in Figure 1 are in proximity (it exists a short path connecting them) while nodes *a* and *b* are not. Instead, structural-based node representations assume that if two nodes have a similar structure, they should have the same representation, e.g., in Figure 1, nodes *c* and *d* are structurally identical even if they are not connected, the same for the nodes *a* and *b*.

Connectivity-based approaches are usually better for the task of edge prediction, while structural-based approaches are more suitable for node classification tasks [5]. This is because the structural representations helps the identification of structural patterns useful for the classification. Moreover, connectivity-based approaches in the case of two disconnected nodes (such as *g* and *h*) may have completely different representations even if, according to a specific classification task, these two nodes should have the same class.

The structural representation learning approaches already proposed in the literature, such as Struc2vec [5] and GraphWave [6], do not scale well on large graphs because their non linear execution time. In addition, as we show in our experimental section for some structural classification tasks, they sometimes present some poor performance. In this work, we propose a structural representation approach, called SparseStruct, that creates for each node a sparse representation that is condensed through a truncated singular value decomposition. The sparse representation is created with an iterative approach exploring the structure of a node, level by level. More specifically, this paper provides the following contributions: (i) we define SparseStruct, a structural representation learning approach with a convergent iterative procedure; (ii) we provide formal proof about the ability of SparseStruct to preserve the structural information of nodes; (iii) we prove that SparseStruct’s execution time is linear on the size of the graph; (iv) from the experimental results, SparseStruct in all the experimental cases is comparable or better than the other approaches considered in terms of performance for

regression and classification tasks.

2. Background

In this section, we first present a set of important notations and concepts that will be used throughout this work.

2.1. Graphs Basics and Vectorial Representation

Let $G = (V, E)$ be a unlabeled undirected graph, or network, where V denotes the set of nodes, and $E \subseteq (V \times V)$ denotes the set of edges connecting the nodes in G . $|V|$ and $|E|$ denote the number of nodes and edges contained in the graph respectively. The neighborhood of any node $u \in V$ is given by $nbr(u) = \{v | (u, v) \in E \text{ or } (v, u) \in E\}$, and $|nbr(u)|$ denotes the size of this neighborhood or the node's degree. The diameter of a graph is commonly defined as the "longest shortest path" between any two nodes of the graph. More formally, the diameter is the length $\max_{(u, v)} d(u, v)$ between any two graph nodes (u, v) , where $d(u, v)$ is the distance between nodes u and v .

The representation of a node u is defined as $X_u \in \mathbb{R}^k$, where k is the desired size of the representation. Finally, the whole representation matrix is defined as $X \in \mathbb{R}^{|V| \times k}$. Graph representation learning methods require a notion of proximity between nodes in order to learn. These methods can, arguably, be classified into two groups based on the proximity notion they use, connectivity-based methods (K-order Proximity), and structure-based methods (Structural Role Proximity) [7].

3. Truncated SVD

Dimensionality reduction approaches are used to reduce the size of a set of features. Truncated SVD [8] is a Linear dimensionality reduction using Singular Value Decomposition to reduce the dimension. Contrary to Principal Component Analysis [9], Truncated SVD does not center the data before computing the singular value decomposition. Such difference makes this approach suitable to work with sparse matrices.

Truncated SVD has been widely applied on text analysis applications [10]. It is known as latent semantic analysis (LSA), and it is applied to term-document matrices. The result of such transformation is a "semantic" space of low dimensionality that contrasts the effects of synonymy and polysemy, i.e. when there are multiple meanings for the same word. Truncated SVD is also used in graphs by transforming the sparse graph adjacency matrix [11]. This produces a connectivity-based graph representations. In our work, we will show how to effectively use the truncated SVD procedure with a newly defined sparse matrix to produce a structural graph representation learning procedure.

The fastest solver for Truncated SVD is based on a randomized approach [12]. The time complexity for this approach is $O(k \times T_{mult} + (n + m) \times k^2)$, where k is the size of the reduced dimensional space, T_{mult} is the execution time of the sparse matrix multiplication with a vector, n is the number of rows in the sparse matrix, and m is the number of columns.

4. SparseStruct Algorithm

In this section, we provide the details of SparseStruct, described in Algorithm 1, that learns a structural node representation. SparseStruct is divided into two steps: **Step 1** (line 2) use an iterative procedure to create a sparse positive integer matrix where each row corresponds to a sparse vectorial representation of a specific node, and **Step 2** (line 3) uses the Truncated SVD to transform the sparse matrix in a smaller dense representation with k dimensions, (k represents the size of the learned output representation). The matrix M (line 3) represents the output of our algorithm.

Intuitively, at each iteration i of the process the algorithm determines the structural information of a generic node u , by considering all the possible structure combinations present in the graph at depth i . All these structures are indexed and used to describe a node's structure in terms of its neighboring nodes structures. The concatenation of all the sparse matrix through a horizontal stacking tells how the structures connected to a node changes step by step with the increase of the exploration depth. This process runs until no more change is detected by the algorithm. For this reason, we called the maximum number of iteration *explorationDepth* (line 11).

More specifically, in this algorithm, we assume that each node of the graph has a unique identifier $id(v)$ from 0 to $|V| - 1$, and such identifier is the index of all the matrices (sparse and not) created in the algorithm. Then the final representation X_u learned for a generic node u is given by $M[id(u), :]$. The core part of Algorithm 1 is the iterative procedure *sparseMatrixGen* that creates the structural representative sparse matrix SM , and it is inspired by the Weisfeiler-Lehman method for graph isomorphism testing [13]. At each iteration i , the *sparseMatrixGen* method assigns to all the identical rows the same progressive identifier with the use of a hash tree index (line 12), and then create a sparse matrix SM^i where the value of the generic cell $SM^i[id(u), j]$ is the number of neighbor nodes of u that are indexed with the identifier j (lines 13-15). Each matrix SM^i has dimension $|V| \times |indexID(SM^i)|$ however its occupied memory space is less than $O(|E|)$. Each generated sparse matrix SM^i inside the while loop is added to the list L (line 17), and after the while loop, all the sparse matrices in L are horizontally stacked to create a unique sparse matrix called SM^{tot} that is the output of the method. The while loop (lines 11-25) iterates until the number of indexes generated (line 12) stops to grow at each iteration, or a maximum number of iterations is reached.

While this approach may, at first glance, seem space and time consuming depending on the number of iteration of the algorithm, in reality, we will show in the time and space complexities section that this is not the case. This is because of the fast convergence of our algorithm.

Our representation approach is able to preserve structural properties of the nodes in the graph (see Theorem 1).

Theorem 1. *Given a graph $G = (V, E)$, for each pair of nodes (u, v) having the same structure (i.e., the graphs extracted from a visit of u and v are isomorphic) the representations $M[id(u), :]$ and $M[id(v), :]$ are the same.*

In the following subsections, we show: (i) the optimality of the stopping criterion, and (ii) the time and space complexity of our proposed approach, that under certain conditions are

Algorithm 1 SparseStruct representation learning algorithm.

```
1: function SparseStruct( $G = (V, E)$ ,  $explorationDepth$ ,  $k$ )
2:    $SM = \text{sparseMatrixGen}(G, \text{explorationDepth})$ 
3:    $M = \text{TruncatedSVD}(SM, k)$ 
4:   return  $M$ 
5: end function
6: function sparseMatrixGen( $G = (V, E)$ ,  $explorationDepth$ )
7:   Initialize a sparse matrix  $SM^0 \in \mathbb{Z}^{|V| \times 1}$  to zero
8:   Initialize an empty list  $L$  of sparse matrices.
9:    $i = 1$ 
10:   $len = 0$ 
11:  while  $i \leq explorationDepth$  do
12:     $index = \text{IndexID}(SM^i)$ 
13:    Initialize a sparse matrix  $SM^i \in \mathbb{Z}^{|V| \times |index|}$  to zero
14:    for all  $(u, v) \in E$  do
15:       $SM^i[id(u), index(SM^{i-1}[id(v), :])] += 1$ 
16:    end for
17:    append  $SM^i$  on  $L$ 
18:    if  $len = |index|$  then
19:      break
20:    else
21:      append  $SM^i$  on  $L$ 
22:       $len = |index|$ 
23:       $i = i + 1$ 
24:    end if
25:  end while
26:   $SM^{tot} = \text{horizontalStack}(L)$ 
27:  return  $SM$ 
28: end function
```

linear on the size of the graph.

4.1. Stopping Criterion

Theorem 1 and Theorem 2 prove that the stopping criteria (line 18) of the function *sparseMatrixGen* in Algorithm 1 is correct and avoids column duplications that do not add information to the generated sparse matrix.

Theorem 2. *In the function *sparseMatrixGen*, in Algorithm 1, for each $i \geq 0$, for any pair of nodes a and b , if $SM^i[id(a), :] \neq SM^i[id(b), :]$ then also $SM^{i+1}[id(a), :] \neq SM^{i+1}[id(b), :]$*

Theorem 3. *Given the stopping iteration s , function *sparseMatrixGen* (in Algorithm 1) stops due to the condition in line 18, the column of the matrix SM^{s-1} can be reordered to be the same of SM^s . This implies that it does not need to continue the iterations since the newly generated sparse matrices will continue to have always the same information of SM^{s-1} .*

4.2. Time and Space Complexities

We describe the time and space complexities of Algorithm 1. The following theorem shows that, assuming the *explorationDepth* a constant, the method *sparseMatrixGen* has space and execution time linear in the number of edges of the graph times the number of iterations. Consequently, considering the $\min(\text{explorationDepth}, s)$ as a constant, our proposed method is linear (in space and time) on the graph size.

Theorem 4. *Let s be the iteration when the function *sparseMatrixGen* (in Algorithm 1) stops due to the condition in (line 18), the time and space complexity of the method *sparseMatrixGen* in Algorithm 1 is $O(|E| \times \min(\text{explorationDepth}, s))$*

The entire algorithm complexity depends on the cost to compute the sparse matrix plus the cost to apply the Truncated SVD on the aforementioned matrix. The following theorem gives an idea of the whole execution cost of the algorithm.

Theorem 5. *Let s be the iteration when the function *sparseMatrixGen* (in Algorithm 1) stops due to the condition in (line 18), the execution time of Algorithm 1 is $O((|V|+|E| \times \min(\text{explorationDepth}, s)) \times k^2)$*

It is important to note that the execution time for the Truncated SVD is a pessimistic upper bound, as mentioned in [7], and, in average the execution time is small. In term of space complexity of the entire approach we have still $O(|v| + |E| \times \min(\text{explorationDepth}, s))$.

5. Experimental Results

In this section we present a set of experiments to empirically verify the capabilities of SparseStruct. We compare our approach with the following six well established techniques: node2vec [14], struc2vec [5], GraphWave [6], GraphSAGE [15], ARGAs [16], and DRNE [17].

To test these approaches, we select a set of small to large-sized real-world graphs. They range from hundreds to thousands of nodes. The chosen graphs are:

- **Facebook348** [18]: This network is one of the ten available ego networks (id 348) of the Stanford ego-Facebook dataset. The network contains 224 nodes and 3,192 edges.
- **Air-traffic networks** [5]: In this set of networks, nodes represent airports and edges indicate the existence of commercial air traffic between them. We will use three different sized air-traffic networks: **Brazilian air-traffic network** (131 nodes and 1,038 edges), **European air-traffic network** (399 nodes and 5,995 edges), and **American air-traffic network** (1,190 nodes and 13,599 edges).
- **Wikipedia** [14]: This network contains the co-occurrence of words appearing in the first million bytes of the Wikipedia dump. The network contains 4,777 nodes and 184,812 edges.

All experiments used an Intel Xeon E5-2620 2.00GHz CPU with 128 GB of RAM memory and an Nvidia GeForce GTX 1080 Ti GPU with 11 GB of dedicated memory. SparseStruct is implemented in python and for all the other methods, we used the Python implementation provided by the authors and, unless otherwise stated, default parameters provided in the code and their documentation.

Table 1 PageRank (PR), HITS Authority/Hub (HITS), Degree Centrality (DC), Eigenvector Centrality (EC), Betweenness Centrality (BC), and Node Clique Number (NCN) centrality measure regression results.

Brazilian air-traffic network												
	PR		HITS		DC		EC		BC		NCN	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Degree	0.016	0.990	0.047	0.962	-	-	0.047	0.962	0.071	0.571	0.076	0.946
node2vec	0.183	0.111	0.229	0.203	0.192	0.137	0.229	0.203	0.141	-0.043	0.280	0.324
struc2vec	0.029	0.968	0.031	0.972	0.029	0.975	0.036	0.972	0.066	0.304	0.064	0.959
GraphWave	0.020	0.978	0.035	0.958	0.021	0.975	0.034	0.956	0.082	0.383	0.089	0.920
GraphSAGE	0.163	0.253	0.209	0.333	0.171	0.275	0.209	0.333	0.136	-0.070	0.258	0.425
ARGA	0.045	0.927	0.048	0.954	0.040	0.949	0.048	0.954	0.052	0.606	0.087	0.933
DRNE	0.016	0.991	0.037	0.974	0.007	0.998	0.037	0.973	0.071	0.569	0.061	0.966
SparseEmb	0.009	0.997	0.008	0.999	0.009	0.998	0.008	0.999	0.049	0.766	0.086	0.933

European air-traffic network												
	PR		HITS		DC		EC		BC		NCN	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Degree	0.020	0.983	0.034	0.973	-	-	0.034	0.973	0.067	0.522	0.113	0.850
node2vec	0.159	0.112	0.190	0.232	0.159	0.166	0.190	0.232	0.104	0.009	0.256	0.280
struc2vec	0.023	0.979	0.030	0.979	0.017	0.989	0.030	0.979	0.051	0.744	0.107	0.868
GraphWave	0.035	0.856	0.036	0.968	0.031	0.873	0.036	0.968	0.044	0.808	0.100	0.879
GraphSAGE	0.087	0.737	0.097	0.796	0.083	0.767	0.097	0.796	0.091	0.279	0.143	0.768
ARGA	0.016	0.991	0.025	0.986	0.015	0.992	0.025	0.986	0.035	0.874	0.096	0.897
DRNE	0.019	0.986	0.026	0.986	0.007	0.998	0.026	0.986	0.059	0.610	0.098	0.886
SparseEmb	0.008	0.997	0.007	0.999	0.007	0.998	0.007	0.999	0.048	0.805	0.068	0.949

American air-traffic network												
	PR		HITS		DC		EC		BC		NCN	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Degree	0.032	0.889	0.054	0.932	-	-	0.054	0.932	0.041	0.127	0.066	0.934
node2vec	0.065	0.554	0.113	0.716	0.278	0.273	0.113	0.716	0.043	0.230	0.129	0.770
struc2vec	0.026	0.921	0.033	0.971	0.011	0.995	0.033	0.971	0.038	-0.168	0.056	0.953
GraphWave	0.018	0.965	0.019	0.989	0.017	0.989	0.019	0.989	0.034	-0.212	0.032	0.983
GraphSAGE	0.040	0.844	0.044	0.966	0.047	0.920	0.044	0.956	0.044	0.216	0.061	0.947
ARGA	0.015	0.976	0.033	0.975	0.018	0.989	0.033	0.975	0.022	0.628	0.058	0.950
DRNE	0.024	0.941	0.031	0.973	0.010	0.973	0.010	0.996	0.034	-0.387	0.039	0.976
SparseEmb	0.010	0.986	0.004	0.999	0.009	0.997	0.004	0.999	0.035	0.553	0.033	0.984

Facebook348												
	PR		HITS		DC		EC		BC		NCN	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Degree	0.060	0.854	0.066	0.932	-	-	0.066	0.932	0.106	0.301	0.103	0.868
node2vec	0.137	0.524	0.148	0.713	0.131	0.647	0.148	0.713	0.117	0.118	0.153	0.705
struc2vec	0.044	0.947	0.047	0.964	0.023	0.988	0.047	0.964	0.098	0.387	0.086	0.908
GraphWave	0.020	0.987	0.027	0.988	0.013	0.995	0.027	0.988	0.090	0.476	0.085	0.910
GraphSAGE	0.161	0.312	0.168	0.624	0.152	0.507	0.168	0.624	0.126	-0.041	0.195	0.537
ARGA	0.032	0.964	0.027	0.990	0.018	0.993	0.027	0.990	0.089	0.508	0.076	0.928
NDRE	0.028	0.976	0.061	0.947	0.004	0.999	0.061	0.947	0.103	0.309	0.100	0.876
SparseEmb	0.037	0.937	0.009	0.999	0.007	0.999	0.009	0.999	0.107	0.313	0.092	0.894

Wikipedia												
	PR		HITS		DC		EC		BC		NCN	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Degree	0.006	0.951	0.010	0.960	-	-	0.010	0.959	0.009	0.001	0.062	0.900
node2vec	0.027	0.420	0.033	0.566	0.028	0.440	0.033	0.566	0.012	-1.94	0.129	0.573
struc2vec	0.007	0.945	0.007	0.976	0.007	0.951	0.007	0.976	0.008	-0.385	0.058	0.915
GraphWave	0.006	0.958	0.005	0.988	0.005	0.974	0.005	0.988	0.008	-1.265	0.051	0.932
GraphSAGE	0.033	0.131	0.044	0.223	0.033	0.138	0.044	0.223	0.022	-1.291	0.170	0.262
ARGA	0.005	0.975	0.005	0.987	0.004	0.978	0.005	0.987	0.006	0.114	0.047	0.944
DRNE	0.020	0.637	0.028	0.693	0.023	0.556	0.027	0.693	0.011	-3.47	0.078	0.845
SparseEmb	0.004	0.977	0.004	0.992	0.004	0.978	0.004	0.994	0.005	0.092	0.042	0.955

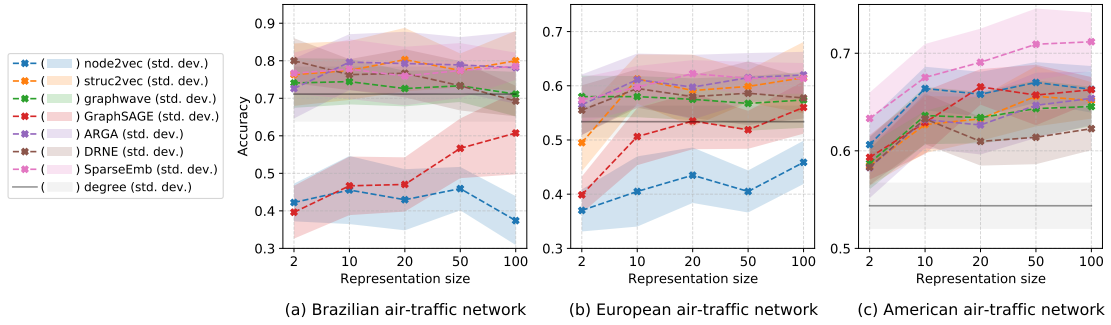


Figure 2: Accuracy (mean and standard deviation among the folds) prediction task vs. vectorial representation size.

5.1. Centrality Measure Regression Experiments

To evaluate the structural information encoded inside the generated representations, we propose a set of regression-based experiments where we use our representations to extract known centrality metrics. We consider the following centrality metrics: PageRank (PR), HITS Authority/Hub, Degree Centrality (DC), Eigenvector Centrality (EC), Betweenness Centrality (BC), and Node Clique Number (NCN).

To do so, we compute all the node representations and network metrics to generate the feature datasets and the ground truths to train and test the regression models. For the regression task, we use Support Vector Machine regression [19], and the optimal hyper-parameters (including kernel type) are selected with a grid-search approach and 10-fold cross-validation. The performance metrics we use are the Root Mean Square Error (RMSE) and the Coefficient of Determination (R^2), obtained as the average value computed over the results of the 10-folds.

In Table 1 are reported the results of each metric for each dataset and for each representation learning approach. As a baseline, we considered the degree of each node since one SM^1 generated by the sparseMatrixGen method in Algorithm 1 contains the degree of each node. Note that the results for the degree in the case of the degree centrality are not reported since two are the same.

From Table 1, SparseStruct is often outperforming all the other representation learning techniques or produce comparable results.

5.2. Classification

We test and compare SparseStruct and all the other considered approach with the node classification task. The node classification task consists in predicting a label for a set of unlabeled nodes. For these experiments, we selected the Brazilian, European, and American air-traffic networks from [5]. In order to test a structure-based classification setting, the datasets have been labeled according to each airport’s activity level, where each airport activity is measured by the total number of airplane landings and takeoffs. Each airport on the datasets has been assigned with one of four possible activity levels. With these levels being the labels for our classification task. In order to assign each label, authors use the quartiles from the empirical

airport activity distribution to split the dataset into four airport groups, each of these groups containing 25% of the total airports. The labels are ordered in increasing activity order, where group 1 contains airports with the least activity, and group 4 contains airports with the most activity. The resulting groups are used to label each node in the datasets. This process has been applied to each dataset separately. Given the representation provided by the different approaches, we used the Extra-Tree classifier [20]. As in the previous experiment, we decide the best model hyper-parameters by a combination of grid search and a 10-fold validation schema.

Figure 2 shows the accuracy (mean and standard deviation among the folds) result with different representation sizes: 2,10,20,50 and 100. Please note that the generated datasets are balanced, so mean accuracy among the fold and the standard deviation are accurate measurements. In Figure 2, we can observe that the SparseStruct is one of the best representation learning approaches in Brazilian and European air-traffic networks. While in the American air-traffic network, the SparseStruct clearly over-performs all the other approaches. In addition, in Figure 2, we can observe in SparseStruct has the desirable property that step by step, the representation size also increases the accuracy increases. This happens until, at a certain size, the performance is not going to change.

6. Conclusions and Future Work

In this paper, we proposed SparseStruct, a large-scale sparse structural representation learning approach for graph nodes. SparseStruct theoretically and empirically preserves the graph structure of each node, and it has a execution time linear in the size of the graph. In our experiments, SparseStruct is often superior or comparable to current state-of-art representation learning approaches on both classification and regression tasks.

Future works is mainly oriented to embed novel features of big data within our framework (e.g., [21, 22]).

Acknowledgements

This work is partially supported by NSERC (Canada) and University of Manitoba.

References

- [1] A. Cuzzocrea, I. Song, Big graph analytics: The state of the art and future research agenda, in: Proceedings of the 17th International Workshop on Data Warehousing and OLAP, DOLAP 2014, Shanghai, China, November 3-7, 2014, ACM, 2014, pp. 99–101.
- [2] A. Campan, A. Cuzzocrea, T. M. Truta, Fighting fake news spread in online social networks: Actual trends and future research directions, in: 2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017, IEEE Computer Society, 2017, pp. 4453–4457.
- [3] M. Joaristi, E. Serra, F. Spezzano, Inferring bad entities through the panama papers network, in: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, 2018, pp. 767–773.
- [4] M. Joaristi, E. Serra, F. Spezzano, Detecting suspicious entities in offshore leaks networks, *Social Network Analysis and Mining* 9 (2019) 62.

- [5] L. F. Ribeiro, P. H. Saverese, D. R. Figueiredo, struc2vec: Learning node representations from structural identity, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 385–394.
- [6] C. Donnat, M. Zitnik, D. Hallac, J. Leskovec, Learning structural node embeddings via diffusion wavelets, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 1320–1329.
- [7] D. Zhang, J. Yin, X. Zhu, C. Zhang, Network representation learning: A survey, *IEEE transactions on Big Data* (2018).
- [8] P. C. Hansen, The truncatedsvd as a method for regularization, *BIT Numerical Mathematics* 27 (1987) 534–553.
- [9] J. Shlens, A tutorial on principal component analysis, *arXiv preprint arXiv:1404.1100* (2014).
- [10] P. D. Turney, Mining the web for synonyms: Pmi-ir versus lsa on toefl, in: *European conference on machine learning*, Springer, 2001, pp. 491–502.
- [11] A. C. Gilbert, J. Y. Park, M. B. Wakin, Sketched svd: Recovering spectral features from compressive measurements, *arXiv preprint arXiv:1211.0361* (2012).
- [12] N. Halko, P.-G. Martinsson, J. A. Tropp, Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions (2009).
- [13] B. L. Douglas, The weisfeiler-lehman method and graph isomorphism testing, *arXiv preprint arXiv:1101.5211* (2011).
- [14] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2016, pp. 855–864.
- [15] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: *Advances in neural information processing systems*, 2017, pp. 1024–1034.
- [16] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, C. Zhang, Adversarially regularized graph autoencoder for graph embedding, *arXiv preprint arXiv:1802.04407* (2018).
- [17] K. Tu, P. Cui, X. Wang, P. S. Yu, W. Zhu, Deep recursive network embedding with regular equivalence, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2357–2366.
- [18] J. Leskovec, A. Krevl, Snap datasets: Stanford large network dataset collection, 2014.
- [19] C. Cortes, V. Vapnik, Support-vector networks, *Machine learning* 20 (1995) 273–297.
- [20] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, *Machine learning* 63 (2006) 3–42.
- [21] A. Cuzzocrea, W. Wang, Approximate range-sum query answering on data cubes with probabilistic guarantees, *J. Intell. Inf. Syst.* 28 (2007) 161–197.
- [22] A. Cuzzocrea, Accuracy control in compressed multidimensional data cubes for quality of answer-based OLAP tools, in: 18th International Conference on Scientific and Statistical Database Management, SSDBM 2006, 3-5 July 2006, Vienna, Austria, Proceedings, IEEE Computer Society, 2006, pp. 301–310.