Internal Report num. 13130

Analysis of the effect of size, type and number of background knowledge bases

Jožef Stefan Institute Version 1 FINAL

Abstract: Task 1.4 aims at assessing progress towards our objective of using network analysis methods to analyse background knowledge networks that are too big to be analysed by SDM methods. We will measure how the number of target variables, imbalances in the distribution of labels, and size and number of background knowledge bases effect the speed and accuracy of our algorithm.
For this purpose, we have developed Python library, Py3plex, which focuses on the visualization and analysis of multilayer networks. The library implements a set of simple graphical primitives supporting intra- as well as inter-layer visualization. It also supports many common operations on multilayer networks, such as aggregation, slicing, indexing, traversal, and more. This work also focuses on how node embeddings can be used to speed up contemporary (multilayer) layout computation. The library's functionality is show cased on both real and synthetic networks.
For huge knowledge networks, we have developed an efficient graph sampler which samples based on a distribution of lengths of random walks, implemented in Numba, offering 15x faster sampling than a Python-native implementation, scaling to graphs with millions of nodes and edges on an of-the-shelf laptop. Within the same study we propose a Symbolic graph embedding (SGE), a symbolic representation learner that is explainable and achieves state-of-the-art performance for the task of node classification and provide evidence that symbolic node embeddings can perform comparably to black-box node embeddings, whilst requiring less space and data.

Document administrative information	
Project acronym:	SDM-Open
Project number:	N2-0078
Deliverable number:	D1.4
Deliverable full title:	Analysis of the effect of size, type and number of background knowledge bases
Document identifier:	SDM-Open D1.4-v1
Lead partner short name:	JSI
Report version:	Version 1, FINAL
Report preparation date:	30/04/2020
Lead author:	Blaž Škrlj
Co-authors:	Nada Lavrač, Jan Kralj
Status:	Final

Introduction

To assess progress towards our objective of using network analysis methods to analyse background knowledge networks that are too big to be analysed by SDM methods, we will measure how the number of target variables, imbalances in the distribution of labels, and size and number of background knowledge bases effect the speed and accuracy of our algorithm.
For this purpose, we have developed Python library, Py3plex, which focuses on the visualization and analysis of multilayer networks. The library implements a set of simple

graphical primitives supporting intra- as well as inter-layer visualization. It also supports many common operations on multilayer networks, such as aggregation, slicing, indexing, traversal, and more. This work also focuses on how node embeddings can be used to speed up contemporary (multilayer) layout computation. The library's functionality is show cased on both real and synthetic networks.
For huge knowledge networks, we have developed an efficient graph sampler which samples based on a distribution of lengths of random walks, implemented in Numba, offering 15x faster sampling than a Python-native implementation, scaling to graphs with millions of nodes and edges on an of-the-shelf laptop. Within the same study we propose a Symbolic graph embedding (SGE), a symbolic representation learner that is explainable and achieves state-of-the-art performance for the task of node classification and provide evidence that symbolic node embeddings can perform comparably to black-box node embeddings, whilst requiring less space and data.

We have published this work in two conference papers, which are added to this report.

Py3plex: A library for scalable multilayer network analysis and visualization

Blaž Škrlj[1,2], Jan Kralj[1], and Nada Lavrač[2,3]

[1] Jožef Stefan International Postgratuate School, Jamova 39, 1000 Ljubljana, Slovenia
[2] Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
[3] University of Nova Gorica, Glavni trg 8, 5271 Vipava, Slovenia
{blaz.skrlj,jan.kralj,nada.lavrac}@ijs.si

Abstract. Real-life systems are commonly represented as networks of interacting entities. While homogeneous networks consist of nodes of a single node type, multilayer networks are characterized by multiple types of nodes or edges, all present in the same system. Analysis and visualization of such networks represent a challenge for real-life complex network applications. The presented Py3plex Python-based library facilitates the exploration and visualization of multilayer networks. The library includes a diagonal projection-based network visualization, developed specifically for large networks with multiple node (and edge) types. The library also includes state-of-the-art methods for network decomposition and statistical analysis. The Py3plex functionality is showcased on real-world multilayer networks from the domains of biology and on synthetic networks.

1 Introduction

Analysis and visualization of complex networks offers novel opportunities to study intractable systems, such as protein interaction networks, transportation networks or social networks [4, 23]. As this vibrant research field offers novel tools at an increasing pace, development of freely available, scalable software resources is becoming a relevant research direction. Despite many existing tools for analysis and visualization of homogeneous networks, i.e. networks with only a single node type and no annotated edges, not many tools consider multilayer (heterogeneous) networks. In multilayer networks, many different types of annotations are taken into account, including e.g., relation-labeled edges, multiple node types etc. Such networks are considered, for example, when multiple layers of biological information (e.g., protein-protein, gene-gene interactions etc.) are available and need to be taken into account when studying diseases [16, 5]. In this work we consider networks as multilayer when they contain at least two types of nodes and/or edges.

This paper presents Py3plex, a new Python library for analysis and visualization of large, heterogeneous (and homogeneous) networks. The visualization suite simplifies displaying of multilayered networks and network communities as well as network embeddings [9]. Further, current version of Py3plex implements a state-of-the-art procedure for converting heterogeneous networks to homogeneous networks [14]. Finally, the library includes a set of subroutines for partition enrichment—the process of learning qualitative explanations relating individual partitions (e.g., communities) to expert-curated domain knowledge [21].

This paper is structured as follows. First, we present the related work on multilayer network analysis, followed by the description of the Py3plex architecture, its comparison with the state-of-the-art approaches, and the proposed multilayer network visualization approach. We showcase visualization performance od Py3plex on selected real-world biological networks, and present the comparison of Py3plex results with the results of the state-of-the-art Pymnet visualization library on synthetic networks. We empirically demonstrate that Py3plex, despite being a lightweight analysis library, offers a novel method for visualization of multilayer networks, performing significantly faster than the current alternatives.

2 Related work

This section presents the state-of-the-art network analysis libraries, relevant to the development of Py3plex. The most common approaches to network analysis can be split into two groups: GUI-based solutions and API-based solutions.

The GUI-based solutions include Cytoscape [19] and Gephi [2]. Cytoscape [19] is one of the largest network analysis projects to date. It supports custom manipulation of the loaded network, and is hence flexible both in terms of network visualization as well as analysis. The Gephi [2] suite offers a similar set of functionalities, yet it is known for better visualization capabilities—a key aspect of the modern network science. The two solutions are mostly used in the final step of a network analysis project, where pre-computed node properties are used as part of the input.

The API-based solutions, which provide programmatic access from popular languages (such as Python, R, JavaScript, C++), are preferred when the entire network analysis and visualization should be carried out in the same environment. The NetworkX library [10] is prevalent for the Python environment, while the igraph library [17] is commonly used among R users. Further, C, and C++ alternatives include SNAP [15] and Boost Graph Library (BGL) [20]. Compared to GUI-based analysis, API-based approaches result in a series of high-level function calls, which generate the desired output. Such approaches are commonly used for analysis when either the networks are large or the number of networks under consideration is high.

The aforementioned approaches focus on homogeneous networks consisting of single node (and edge) types. On the other hand, recent advances in multilayer network analysis prove that additional information associated with node and edge types provides insights regarding network structure and dynamics. Multilayer networks can be represented as higher order tensors [7], encoding inter- as well as intra-layer connections of varying intensity. In this paper we focus on state-of-the-art implementations of this formalism, their functionality and some of their drawbacks. The currently used libraries for visualization and analysis of multilayer networks include:

- libraries for the Python environment Pymnet[4] [13] and MultinetX[5] [1], and
- library for the R environment Muxviz[6] [6].

[4] http://www.mkivela.com/pymnet/

[5] https://github.com/nkoub/multinetx

[6] http://muxviz.net/

Detailed analysis of these approaches is presented in Section 4, where they are compared to the proposed Py3plex library.

3 Py3plex library architecture

This section explains the proposed Py3plex library's architecture, followed by the description of individual components and subroutines.

A high-level organization of the Py3plex library is shown in Figure 1. The library includes methods for parsing and converting graphs from and to various formats, while the core library includes three modules:

- **The visualization module** consists of different subroutines used for network visualization of multilayer and single layer networks. Detailed description of the in-house developed multilayer visualization is given in Section 5.
- **The wrappers module** is implemented as follows. As many procedures are given as standalone executables, Py3plex offers a set of wrapper subroutines, useful for calling external state-of-the-art algorithms, for example, highly optimized network embedding routines [9] as well as the InfoMap community detection algorithm [18] .
- **The algorithms module** includes implementations of many commonly used algorithms, such as: Louvain community detection [3], node ranking, network statistics and the recently introduced network topology enrichment [21].

All the modules are built in an extensible manner, allowing for new routines to be easily added. As Py3plex is built on top of the NetworkX [10], network operations can include any of the methods primarily designed for homogeneous networks. This functionality provides flexibility when implementing novel multilayer network analysis algorithms.

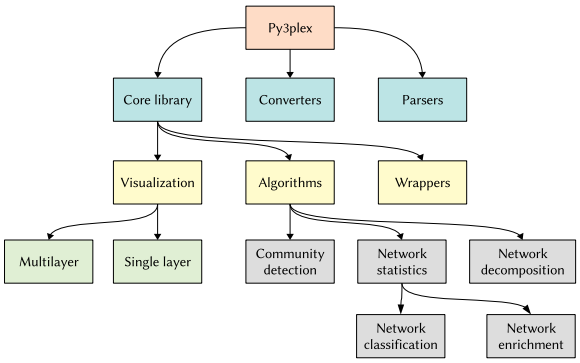


Fig. 1: The Py3plex library architecture. The library is organized in hierarchical manner — a set of core data structures (blue) is used in algorithms for visualization and analysis of multialyer networks.

4 Py3plex comparison to other libraries

In this section we compare the functionality of different multilayer network analysis libraries. Each library offers some functionality that is not available in other libraries, hence no library is exhaustive. We evaluate different aspects, ranging from computational efficiency to general richness in terms of various analysis functions. The results are summarized in Table 1.

To ensure sufficient speed of execution, Pymnet [13], Py3plex and MultinetX [1] include subroutine implementations in C (via Cython), but also Numpy [22] and Scipy [12] libraries for optimized linear algebra-based computation. The R-based MuxViz [6] uses the Octave library [8] for efficient array-based operations.

In terms of visualization, all compared packages include at least some form of network visualization. Apart from 2D layouts, The MuxViz and Pymnet libraries also support 3D layouts. As discussed in Section 5, Py3plex offers a new approach to multilayer network visualization, whereas MultinetX, for example, displays the supra-adjacency matrix [7] of the network in a block-diagonal manner. The multitude of different multilayer network visualization applications indicates no single type of visualization is optimal for a given task, as each visualization is optimal for a particular range of network size and density.

Apart from MultinetX, all other libraries include different methods for network aggregation and decomposition. A similar compendium of functionality is offered by Pymnet and MuxViz. All packages offer intuitive access to a network's supra-adjacency matrix, and hence provide many opportunities for implementation of computationally efficient multilayer network analysis algorithms.

Table 1: Comparison of multilayer network analysis libraries.

	Py3plex	Pymnet [13]	MuxViz [6]	MultinetX [1]
Core features				
Programming language	Python 3 and C (via Numpy and Cython)	Python 3 and C (via Numpy)	R	Python 3 and C (via Numpy)
Basic statistics	✓	✓	✓	✓
Visualization of large networks	✓	-	✓	-
Visualization in 3D	-	✓	-	✓
Aggregation/decomposition	✓	✓	✓	-
Random graph generators	✓	✓	✓	✓
Adjacency matrix manipulation	✓	✓	✓	✓
Tensorial manipulation	✓	✓	✓	✓
Additional features				
Machine learning for multilayer networks	✓	-	-	-
GUI version	-	-	✓	-
Community detection	✓	-	✓	-
Isomorphisms	✓	✓	-	-
Node ranking	✓	✓	✓	✓
Semantic topology enrichment	✓	-	-	-
Temporal networks	-	-	✓	✓
Network embedding functionality	✓	-	-	-

Py3plex also leverages full functionality of the recently developed HINMINE [14] methodology for heterogeneous network decomposition. As standard aggregation schemes yield a homogeneous network by summing over edges in individual layers, HINMINE-based aggregation is based on frequency of relation-annotated directed paths of length two between the target node type. Compared with standard aggregation, HINMINE is thus suitable for situations where domain-knowledge was used for annotating the network edges. Further, Py3plex also includes basic aggregation subroutines. All libraries include methods for obtaining quick insights into a network's structure. The Pymnet and Py3plex are currently the only libraries supporting different isomorphism algorithms for evaluating network similarity.

In terms of other functionalities, we observe the following. MuxViz also supports GUI-based analysis, as, once installed, it can be executed locally within the browser as a standalone software. To our knowledge MuxViz is the only alternative for GUI-based multilayer network exploration.

We observe Pymnet offers one of the most intuitive API interfaces for manipulation of tensor representations of multilayer networks, such as slicing, indexing etc. In comparison, MultinetX and MuxViz offer similar functionality, whereas Py3plex operates on attribute-rich list-based representations of a network and is not necessarily suitable for all multilayer slicing tasks.

Apart from network analysis and visualization capabilities, Py3plex is the only library which also supports various forms of learning from multilayer networks. It provides the methodology for semantic enrichment on complex networks. The main objective in this emerging field is to associate previously known domain knowledge with topological structures of a network. For example, the functional characterization of a network's communities can only be assessed using curated domain knowledge. We refer to the use of such knowledge to understand network-topological properties as network enrichment. Currently, Py3plex is the only library that supports both rule-based enrichment as well as standard Fisher's exact test-based enrichment, both recently introduced as part of the Community-based Semantic Subgroup Discovery methodology [21]. Further, classification using Personalized PageRank vectors is also supported [14].

Current implementation of Py3plex contains wrapper routines for widely used network embedding algorithms, such as Node2vec and similar [9]. Such functionality is not offered in any other libraries.

Overall, Py3plex thus offers novel algorithms for understanding network structure, an intuitive interface for manipulation of multilayer networks as well as network decomposition and aggregation, however, the primary features of the current version of Py3plex are primarily network visualization and spatially efficient network manipulation (linear in terms of nodes and edges). This functionality offers additional state-of-the-art performance, not observed in the other libraries. We prove this claim in the following sections, where we first present the visualization algorithm, followed by empirical evaluation of its performance on a set of artificial random multilayer networks.

5 Multilayer network visualization

One of the key contributions of this paper is a novel method for visualization of multilayer networks. In this section we present the proposed visualization method and its implementation. Next, we evaluate the method's performance on a series of random networks, where we also compare the results with Pymnet's network visualization capabilities.

5.1 Visualization methodology and implementation

The implementation of the proposed layout algorithm operates in thre main steps, described below in detail.

- **Intra-layer layout computation**. First, subnetworks consisting of same-typed nodes and edges between them are considered. For each node type, we compute a fast, force-directed layout on the single-type subnetwork. This results in (x, y) coordinate pairs for individual nodes which are normalized so that both coordinates are between 0 and 1.
- **Multilayer network drawing**. In the second step, the actual coordinates of each point are then computed by separating each consecutive layer from the preceding layer by exactly a single coordinate unit on both axes. Consequently, nodes are drawn around a diagonal line based on their types. As nodes in different layers are separated by a coordinate unit, each subnetwork corresponding to different node type is drawn in a separate region—this ensures no overlap between different layers is possible, and that each layer includes a network with its own local organization, independent of the inter-layer connections.
- **Inter-layer edge drawing**. The final step involves drawing of inter-layer edges, which are represented as arches connecting different nodes. One of the main problems we faced during the implementation of this step is edge positioning and parameterization, as there are many different options for drawing an arch between two nodes. We consider three possible scenarios in terms of inter-layer edge drawing:
  1. edges are only on the upper part of the layer diagonal,
  2. edges are only on the bottom part of the layer diagonal,
  3. edges are on both sides of the diagonal projection.

To further explain the inter-layer edge drawing step of Py3plex, We first discuss how edges are drawn when only the upper part is considered, and continue with the extension to bottom-only and both parts of the diagonal projection.

An arch connecting nodes $A = (x_1, y_1)$ and $B = (x_2, y_2)$ is drawn through an artificially introduced node $C = (x_f, y_f)$, which lies between A and B at the desired part of the diagonal. Point C is calculated by taking the midpoint between A and B and scaling its y coordinate by a factor of τ (a parameter of the visualization method) from the line.

Once A, B and C are obtained, the points A and B are connected by a parabolic arch that passes through all three points. Even though the current implementation constructs inter-layer edges using interpolation over three points, Py3plex supports adding an arbitrary number of intermediary points, in which case cubic arch interpolation is used to

produce the line between A and B. In our experiments, however, we show that even a single intermediary point allows for clear visualization.

The proposed setting parameterized with τ offers simple manipulation of inter-layer edges, as

1. when $\tau > 1$, the arch will be located above the diagonal,
2. when $\tau = 1$, the arch will be a line between the two nodes (a third point on $f(x)$),
3. when $\tau < 1$, the arch will be located below the diagonal.

We further propose a heuristic for automatically determining an arch's position, i.e. whether an arch should be located above or below the diagonal. We achieve this as follows. Given A and B as defined in the previous paragraph, the arch is located above the diagonal if and only if $int(c) > c$, where $int(c)$ represents the integer representation of coordinate c. Integer-based rounding works, as layers are separated by exactly a single coordinate unit. All operations for arch computation, scaling and transformation are vectorized for better performance. Should the network appear incomprehensible, natural logarithms are applied to node representations—filled circles based on individual node degrees—which simplifies visualization of denser networks

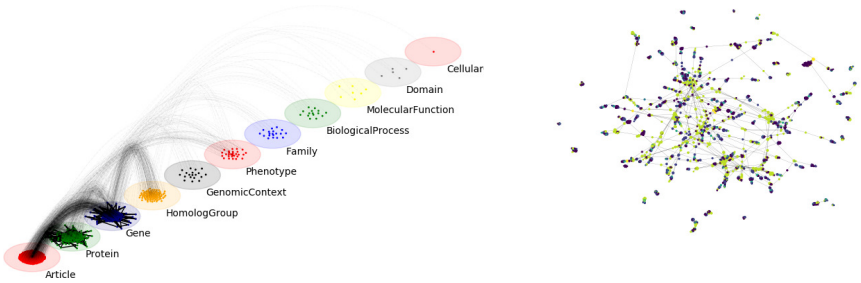
Individual, single-layer networks are derived from the existing NetworkX graph library [10] hence the rich set of functionalities available in NetworkX can be used. Py3plex provides the means to customize majority of network properties, including edge shapes and colors, individual layer layouts, node sizes and colors, and overall network organization. Computationally expensive operations are vectorized using the Numpy numeric library [22]. The Barnes-Hut approximation of the force-directed layout algorithm implementation, which was transpiled from Python to C, is used as the default option during visualization [11]. Py3plex can easily visualize more than ten layers with tens of thousands of nodes. Compared to existing solutions, diagonal projection of multiple layers enables visualization using standard layout algorithms, with additional specification of inter-layer edges.

Example visualization using the proposed Py3plex library is shown in Figure 2, where an alternative visualization using a single layer force-directed layout of the whole network is shown for comparison.

5.2 Comparison of Py3plex and Pymnet visualization performance

In this section we present the results for the times needed to obtain a visualization of networks of different complexities. Even though Pymnet does not support the diagonal projection and Py3plex does not support the default 3D linear projections, both methods are useful

Note that individual node types are drawn along the diagonal axis. Edges can exist either on the intra-layer as well as inter-layer levels. Curves, representing inter-layer edges are adapted based on the distance between the two layers under consideration. Such visualization offers quick insights into inter- and intra-layer dynamics, which can not be detected in hairball (spring-based, single-layered) plots.for visualization of different aspects of multilayer networks. The main aim of this section is to demonstrate that Py3plex offers better support for visualization of larger networks.



(a) Multilayer diagonal projection (b) Single layer hairball plot

Fig. 2: Comparison between a multilayer visualization and single layer (hairball) layout supported in Py3plex. The proposed diagonal projection is shown in subfigure (a), and the hairball plot in (b). The inter-layer connectivity is more clearly expressed in (a) than in (b). Further, organization between the nodes of the same type can not be observed in subfigure (b), as the layout algorithm considers all of the connections. Py3plex supports both visualization styles.

The experimental setting for this task was designed as follows. Random multilayer Erdős-Rényi (ER) networks were generated using the Pymnet [13] library. Such random networks are parameterized using parameter N corresponding to the total number of nodes, parameter L corresponding to the number of layers, and parameter p corresponding to the re-wiring probability. We generated the networks in the following parameter ranges:

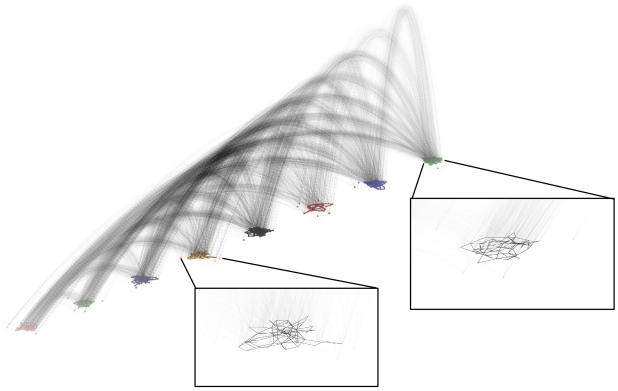
$p \in \{0.05, 0.1, 0.2, 0.3\}$,
$N \in \{5, 10, 20, 50, 80, 100\}$,
$L \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

Once generated, the time needed for visualization was recorded. We compared visualization times of Pymnet and Py3plex, as the remaining Python-based alternative, MultinetX, does not support drawing of coupled edges.

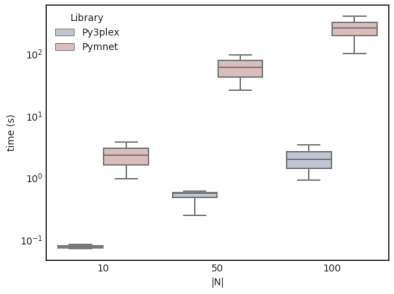
We show the final results of our experiment in the form of box plots, where $|N|$ or $|E|$ is plotted on the x-axis and the time needed is plotted on the y-axis (Figure 3). Networks with more than 100 nodes were not considered for this benchmark, as it took Pymnet more than two hours for visualization. Nonetheless, Py3plex was able to visualize a network with $|N| = 4{,}000$ and $|E| = 18{,}600$ under two hours, even though the obtained network is not necessarily useful for visualization purposes. The machine used for benchmark testing was an off-the-shelf Lenovo y510p laptop.

6 Conclusions and further work

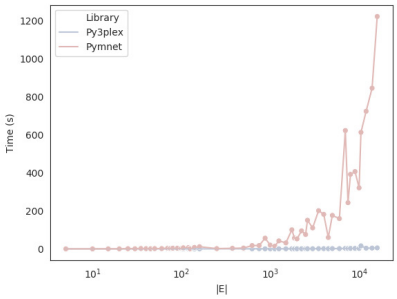
We have developed and implemented Py3plex, a network analysis and visualization library focused on multilayer networks. Apart from most of the functionality offered by the state-of-the-art libraries, Py3plex introduces a novel visualization, suitable for larger



(a) Py3plex-based visualization of a random network with 6,235 edges and eight layers. The network can be explored interactively (inset images).



(b) Visualization time with respect to the number of nodes.



(c) Visualization time with respect to the number of edges.

Fig. 3: Visualization time comparison on random ER network. Even though initial visualization does not necessarily offer intuitive representation of a given network (a), Py3plex offers interactive exploration of the network, where users can zoom-in into parts of interest—as an example, see the inset images in (a). The time Pymnet needed to visualize the network does not appear linear in terms of the number of nodes (b) and edges (c), hence networks with more than 100 nodes were not considered for this benchmark.

multilayer networks, as to our knowledge, there currently do not exist any methods that can visualize networks composed of thousands of nodes along with their intra- as well as inter-layer organization. The Py3plex is suitable for the development of novel algo-

rithms, as it offers fast lookup and indexing routines, which scale to networks consisting of multiple layers with hundreds of thousands of edges.

As the Py3plex analysis suite is by no means exhaustive, further work includes the implementation of network dynamics analysis methods as well as the more efficient, GPU-based random samplers. Further, as Py3plex, apart from offering novel visualization capabilities, aims to bridge the gap between machine learning approaches and complex networks, it does not yet include extensive tensor manipulation, unfolding, and construction routines offered in e.g., the Pymnet library. Finally, as part of the further work we believe Py3plex should be tested on more non-synthetic networks.

7 Availability

Py3plex as well as all datasets used for this paper are freely accessible at https://github.com/SkBlaz/Py3plex, where minimal working examples of the Py3plex functionality are also offered.

8 Acknowledgements

This work was financially supported by the Slovenian Research Agency (ARRS) grants *HinLife: Analysis of Heterogeneous Information Networks for Knowledge Discovery in Life Sciences* (J7-7303) and *Semantic Data Mining for Linked Open Data* (financed under the ERC Complementary Scheme, N2-0078).

References

1. Amato, R., Kouvaris, N.E., San Miguel, M., Díaz-Guilera, A.: Opinion competition dynamics on multiplex networks. New Journal of Physics **19**(12) (2017)
2. Bastian, Mathieu and Heymann, Sebastien and Jacomy, Mathieu and others: Gephi: An open source software for exploring and manipulating networks. International AAAI Conference on Web and Social Media Third International AAAI Conference on Weblogs and Social Media **8**(2009), 361–362 (2009)
3. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), P10,008 (2008)
4. Boccaletti, S., Bianconi, G., Criado, R., del Genio, C., Gmez-Gardees, J., Romance, M., Sendia-Nadal, I., Wang, Z., Zanin, M.: The structure and dynamics of multilayer networks. Physics Reports **544**(1), 1 – 122 (2014)
5. Cho, D.Y., Kim, Y.A., Przytycka, T.M.: Chapter 5: Network biology approach to complex diseases. PLOS Computational Biology **8**(12), 1–11 (2012)
6. De Domenico, M., Porter, M.A., Arenas, A.: MuxViz: A tool for multilayer analysis and visualization of networks. Journal of Complex Networks **3**(2), 159–176 (2015)
7. De Domenico, M., Solé-Ribalta, A., Cozzo, E., Kivelä, M., Moreno, Y., Porter, M.A., Gómez, S., Arenas, A.: Mathematical formulation of multilayer networks. Physical Review X **3**(4) (2013)
8. Eaton, J.W., Bateman, D., Hauberg, S.: GNU Octave version 3.0. 1 manual: a high-level interactive language for numerical computations. SoHo Books (2007)

9. Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pp. 855–864. ACM, New York, NY, USA (2016)
10. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy) (2008)
11. Jacomy, M., Venturini, T., Heymann, S., Bastian, M.: ForceAtlas2, A continuous graph algorithm for handy network visualization designed for the Gephi software. PloS One **9**(6), e98,679 (2014)
12. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–). URL http://www.scipy.org/
13. Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J.P., Moreno, Y., Porter, M.A.: Multilayer networks. Journal of Complex Networks **2**(3), 203–271 (2014)
14. Kralj, J., Robnik-Šikonja, M., Lavrač, N.: HINMINE: Heterogeneous Information Network Mining with Information Retrieval Heuristics. J. Intell. Inf. Syst. **50**(1), 29–61 (2018)
15. Leskovec, J., Sosič, R.: Snap: A general-purpose network analysis and graph-mining library. ACM Trans. Intell. Syst. Technol. **8**(1), 1:1–1:20 (2016)
16. Milano, M., Guzzi, P.H., Cannataro, M.: HetNetAligner: A Novel Algorithm for Local Alignment of Heterogeneous Biological Networks. In: Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB '18, pp. 598–599. ACM, New York, NY, USA (2018)
17. Nepusz, G., Csárdi, G.: The igraph software package for complex network research. Complex Systems **1695**(5), 1–9 (2006)
18. Rosvall, M., Axelsson, D., Bergstrom, C.T.: The map equation. The European Physical Journal Special Topics **178**(1), 13–23 (2009)
19. Shannon, P.: Cytoscape: A software environment for integrated models of biomolecular interaction networks. Genome Research **13**(11), 2498–2504 (2003)
20. Siek, J.G., Lee, L.Q., Lumsdaine, A.: Boost Graph Library: The User Guide and Reference Manual. Pearson Education (2001)
21. Škrlj, B., Kralj, J., Vavpetič, A., Lavrač, N.: Community-based semantic subgroup discovery. In: A. Appice, C. Loglisci, G. Manco, E. Masciari, Z.W. Ras (eds.) Proceedings of New Frontiers in Mining Complex Patterns, pp. 182–196. Springer International Publishing (2018)
22. Walt, S.v.d., Colbert, S.C., Varoquaux, G.: The numpy array: a structure for efficient numerical computation. Computing in Science & Engineering **13**(2), 22–30 (2011)
23. Wang, Z., Wang, L., Szolnoki, A., Perc, M.: Evolutionary games on multilayer networks: a colloquium. The European Physical Journal B **88**(5), 124 (2015)

Symbolic Graph Embedding using Frequent Pattern Mining

Blaž Škrlj[1,2], Nada Lavrač[2,1,3], and Jan Kralj[2]

[1] Jožef Stefan International Postgraduate School
[2] Jožef Stefan Institute, Slovenia
[3] University of Nova Gorica, Slovenia

Abstract. Relational data mining is becoming ubiquitous in many fields of study. It offers insights into behaviour of complex, real-world systems which cannot be modeled directly using propositional learning. We propose Symbolic Graph Embedding (SGE), an algorithm aimed to learn symbolic node representations. Built on the ideas from the field of inductive logic programming, SGE first samples a given node's neighborhood and interprets it as a transaction database, which is used for frequent pattern mining to identify logical conjuncts of items that co-occur frequently in a given context. Such patterns are in this work used as features to represent individual nodes, yielding interpretable, symbolic node embeddings. The proposed SGE approach on a venue classification task outperforms shallow node embedding methods such as DeepWalk, and performs similarly to metapath2vec, a black-box representation learner that can exploit node and edge types in a given graph. The proposed SGE approach performs especially well when small amounts of data are used for learning, scales to graphs with millions of nodes and edges, and can be run on an of-the-shelf laptop.

Keywords: Graphs, machine learning, relational data mining, symbolic learning, embedding

1 Introduction

Many contemporary databases are comprised of vast, linked and annotated data, which can be hard to exploit for various modeling purposes. In this work, we explore how learning from heterogeneous graphs (i.e. heterogeneous information networks with different types of nodes and edges) can be conducted using the ideas from the fields of symbolic relational learning and inductive logic programming [16], as well as contemporary representation learning on graphs [2].

Relational datasets have been considered in machine learning since the early 1990s, where tools such as Aleph [27] have been widely used for relational data analysis. However, recent advancements in deep learning, a field of subsymbolic machine learning, which allows for learning from relational data in the form of graphs, was shown as useful for many contemporary relational learning tasks at scale, including recommendation, anomaly detection and similar [25]. The

state-of-the-art methodology exploits the notion of *node embeddings*—nodes, represented using real-valued vectors. As such, node embeddings can be simply used with propositional learners such as e.g., logistic regression or neural networks. The node embeddings, however, directly offer little to no insight into connectivity patterns relevant for representing individual nodes.

In this work we demonstrate that symbolic pattern mining can be used for learning *symbolic node embeddings* in heterogeneous information graphs. The main contributions of this work include:

1. An efficient graph sampler which samples based on a distribution of lengths of random walks, implemented in Numba, offering 15x faster sampling than a Python-native implementation, scaling to graphs with millions of nodes and edges on an of-the-shelf laptop.
2. Symbolic graph embedding (SGE), a symbolic representation learner that is explainable and achieves state-of-the-art performance for the task of node classification.
3. Evidence that symbolic node embeddings can perform comparably to black-box node embeddings, whilst requiring *less* space and data.

This paper is structured as follows. We first discuss the related work (Section 2), followed by the description of the proposed approach (Section 3), its computational and spatial complexity (Section 4), and its empirical evaluation (Section 5). We finally discuss the obtained results and potential further work (Section 6).

2 Related work

Symbolic representation learning has already been considered in the early 1990s in the inductive learning community, when addressing multi-relational learning problems through the so-called *propositionalization* approach [16]. The goal of propositionalization is to transform multi-relational data into real-valued vectors describing the individual training instances, that are a part of a relational data structure. The values of the vectors are obtained by evaluating a relational feature (e.g., a conjunct of conditions) as true (value 1) or false (value 0). For example, if all conditions of a conjunct are true, the relational feature is evaluated as true, resulting in value 1, and gets value 0 otherwise. We next discuss the approaches which were most influential for this work. The in-house developed Wordification [22] explores how relational databases can be unfolded into bags of relational words, used in the same manner as done in the area of natural language processing via Bag-of-words-based representations. Wordification, albeit very fast, can be spatially expensive, and was designed for SQL-based datasets. Our work was also inspired by the recently introduced HINMINE methodology [14], where Personalized PageRank vectors were used as the propositionalization mechanism. Here, each node is described via its probability to visit any other node, thus, a node of a netwrk is described using a distribution over the remainder of the nodes. Further, propositionalization has recently been explored

in combination with artificial neural networks [8], and as a building block of deep relational machines [6].

Frequent pattern mining is widely used for identifying interesting patterns in real-world transaction databases. Extension of this paradigm to graphs was already explored [12], using the Apriori algorithm [1] for the pattern mining. In this work we rely on the efficient FP-Growth [4] algorithm, which employs more structured counting compared to Apriori using fp-trees as the data structure. Frequent pattern mining is commonly used to identify logical patterns which appear above a certain e.g., frequency threshold. Efficiently mining for such patterns remains a lively research area on its own, and can be scaled to large computing clusters [11].

The proposed work also explores how a given graph can be sampled, as well as embedded efficiently. Many contemporary node representation learning methods, such as node2vec [9], DeepWalk [23], PTE [28] and metapath2vec [7], exploit such ideas in combination with e.g., the skip-gram model in order to obtain low-dimensional embeddings of nodes. Out of the aforementioned methods, only metapath2vec was adapted specifically to operate on *heterogeneous information networks*, i.e. graphs with additional information on node and edge types. It samples pre-defined meta paths, yielding type-aware node representations which serve better for classifying e.g., different research venues to topics.

Heterogeneous (non-attributed) graphs are often formalized as RDF triplets. Relevant methods, which explore how such triplets can be embedded are considered in [5], as well as in [24]. The latter introduced RDF2vec, a methodology for direct transformation of a RDF database to the space of real-valued entity embeddings. Understanding how such graphs can be efficiently sampled, as well as embedded into low-dimensional, real-valued vectors is a challenging problem on its own.

3 Proposed SGE algorithm

In this section we describe Symbolic Graph Embedding (SGE), a new algorithm for symbolic node embedding. The algorithm is summarized in Figure 1. The algorithm consists of two basic steps. First, for each node in an input graph, the neighborhood of the node is sampled (Section 3.2). Next, the patterns, emerging from the walks around a given node are transformed into a set of features whose values describe the node (Section 3.3). In this section, we first introduce some basic definitions and then explain both steps of SGE in more detail.

3.1 Overview and definitions

We first define the notions of a graph as used in this work.

Definition 1 (Graph). *A* graph *is a tuple* $G = (N, E)$*, where* N *is a set of nodes and* E *is a set of edges. The elements of* E *can be either subset* N *of size* 2 *(e.g.,* $\{n_1, n_2\} \subseteq N$*), in which case, we say the graph is* undirected.

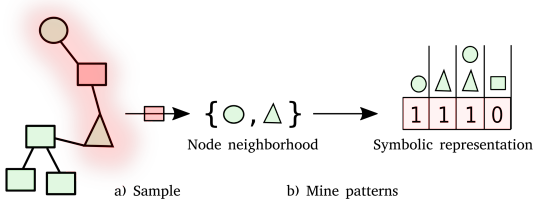


Fig. 1. Schematic represenatation of SGE. The red square's neighborhood (red highlight) is first used to construct symbolic features, forming a propositional graph representation (part a of the figure). The presence of various symbolic patterns (FP) is recorded and used to determine feature vectors for individual nodes (part b of the figure). The obtained representation can be used for subsequent data analysis tasks such as classification or visualization.

Alternatively, E can consist of ordered pairs of elements from N (e.g., $(n_1, n_2) \in N \times N$) – in this case, the graph is directed.

In this work we focus on directed graphs, yet the proposed methodology can also be extended to undirected ones. In this work we also use the notion of a *walk*.

Definition 2 (Walk). *Given a directed graph $G = (N, E)$, a* walk *is any sequence of nodes $n_1, n_2 \ldots, n_k \in N$ so that each pair n_i, n_{i+1} of consecutive nodes is connected by an edge, i.e. $(n_i, n_{i+k}) \in E$.*

Finally, we define the notion of node embedding as used throughout this work.

Definition 3 (Symbolic node embedding). *Given a directed graph $G = (N, E)$, a d-dimensional node embedding of graph G is a matrix $\mathcal{M}$ in a vector space $\mathbb{R}^{|N| \times d}$, i.e. $\mathcal{M} \in \mathbb{R}^{|N| \times d}$. Such embedding is considered* symbolic*, when each column represents a symbolic expression, which, when evaluated against a given node's neighborhood information, returns a real number representing a given node.*

We first discuss the proposed neighborhood sampling routine, followed by the description of pattern learning as used in this work.

3.2 Sampling node neighborhoods

Sampling a given node's local and global neighborhoods offers insights into connectivity patterns of the node with respect to the rest of the graph. Many contemporary methods resort to node neighborhood sampling for obtaining the node co-occurrence information. In this work we propose a simple sampling scheme

which produces a series of graph walks. The walks can be further used for learning tasks—in this work, we use them to produce node representations. Building on recent research ideas [17, 18], the proposed scheme consists of two steps: selection of walk sampling distribution and sampling. We first discuss the notion of distribution-based sampling, followed by the implemented sampling scheme.

Distribution-based sampling Our algorithm is based on the assumption that when learning a representation of a given node, nodes at various distances from the considered node are relevant. In order to use the information on neighborhood nodes, we sample several random walks starting at each node of the graph. Because real-world graphs are diverse, it is unlikely that the same sampling scheme would suffice for arbitrary graphs. To account for such uncertainty, we introduce the notion of *walk distribution vector*, a vector describing how many walks of a certain length shall be sampled. Let $w \in \mathbb{R}^s$ denote a vector of length s (a parameter of the approach). The i-th value of the vector corresponds to the proportion of walks of length i that are to be sampled. Note that the longest walk that can occur is of length s. For example consider the following vector w of length $s = 4$, $w = [0.2, 0, 0.5, 0.3]$. Assuming we sample e.g, 100 random walks, 20 walks will be of length one, zero of length 2, 50 of length three and 30 of length four. As w represents a probability distribution of different walk lengths to be sampled, $\sum_{i=1}^{s} w_i = 1$ must hold.

Having defined the formalism for describing the number of walks of different lengths, we have yet to describe the following two aspects in order to fully formalize the proposed sampling scheme: how to parametrize w and walk efficiently?

How to parametrize w We next discuss the considered initialization of the probability vector w. We attempt to model such vector by assuming a prior *walk length* distribution, from which we first sample ϕ samples—these samples represent different random walk *lengths*. In this work, we consider Uniform walk

Algorithm 1: Order-aware random walker.

Data: A graph $G = (N, E)$
Parameters : Starting node n_i, walk length s

$c \leftarrow n_i$;
$\delta \leftarrow 0$;
$\mathcal{W} \leftarrow$ multiset;
for $\alpha \in [1 \dots s]$ **do**
  $o := \text{Uniform}(N_G(c))$; ▷ `Select a random node.`
  $\mathcal{W} \leftarrow \mathcal{W} \cup (o, \alpha)$; ▷ `Store visited node.`
  $c \leftarrow o$;
end

Result: A random walk $\mathcal{W}$

length distribution, where the i-th element of vector w is defined as $\frac{1}{s}$, where s represents the length of w (maximum walk length).

The considered variant of graph sampling procedure does not take into account node or edge types. One of the purposes of this work is to explore whether such naïve sampling—when combined with symbolic learning—achieves good performance. The rationale for not exploring how to incorporate node and edge types is thus twofold: First, we explore whether symbolic learning, as discussed in the next section, detects heterogeneous node patterns on its own, as the node representations are discrete and could, as such, provide such information. Further, as exact node information is kept intact, and each node can be mapped to its type, the node types are implicitly incorporated. Next, we believe that by selecting the appropriate prior walk distribution w, node types can be to some extent taken into account (yet this claim depends largely on a given graph's topology).

How to walk efficiently An example random walker, which produces walks of a given length (used in this work) is formalized in Algorithm 1. Here, we denote with $N_G(n_i)$ the neighbors of the i-th node. The Uniform$(N_G(c))$ represents a randomly picked neighbor of a given node c, where picking each neighbor is equiprobable. We mark such picked node with o. Note that the walker is essentially a probabilistic depth-first search. The algorithm returns a list of tuples where each tuple (o, α) contains both the visited node o and the step α at which the node was visited. On line 6, we append to a current walk a tuple, comprised of a certain node and the overall walk length, it is a part of. Note that such inclusion of node IDs is suitable in a learning setting, where e.g., part of the graph's labels are not known and are to be predicted using the remainder of the graph. Even though inclusion of such information might seem redundant, we would like to remind the reader that the presented algorithm represents only a single random walker for a single walk length. In reality, multiple walkers yielding walks of different lengths are simulated, since including such positional (walk length) information can be beneficial for the subsequent representation learning step. The proposed Algorithm 1 represents a simple random walker. In practice, thousands of random walks are considered. As discussed, their lengths are distributed according to w. In theory, one could learn the optimal w by using e.g., stochastic optimization, yet we explore a different, computationally more feasible approach for obtaining a given w. We next discuss the notion of symbolic pattern mining and final formalization of the proposed SGE algorithm.

3.3 Symbolic pattern mining

In the previous section we discussed how a given node's neighborhood can be efficiently sampled. In this section we first discuss the general idea behind forming node representations, followed by a description of the frequent pattern mining algorithms employed.

Forming node representations Algorithm 1 outputs a multiset comprised of nodes, represented by (node id, walk order) tuples. Such multisets are in the following discussion considered as *itemsets*, as this is the terminology used in [4]. In the next step of SGE, we use the itemsets to obtain individual node representations. We first give an outline of this step, and provide additional details in the next section. The set of all nodes is considered as a transaction database, and the itemsets comprised of node id and walk order are used to identify frequent patterns (of tuples). Best patterns, selected based on their frequency of occurrence, are used as *features*. The way of determining the best patterns is approach specific, and is discussed in the following paragraphs. Feature values are determined based on the pattern identification method, and are either real-, natural- or binary-valued. Intuitively, they represent the presence of a given node pattern in a given node's neighborhood.

Frequent pattern mining We next discuss the frequent pattern mining approaches explored as part of SGE. The described approaches constitute the *findPatterns* method discussed in the next section. For each node, a multiset of (node ID, walk length) tuples is obtained. In each of the described approaches, the result is a transformation of the set of multisets, describing the network nodes, into a set of feature vectors describing these nodes.

Relational BoW. This paradigm leverages the Bag-of-words (BOW) constructors widely known in natural language processing [31]. Here, the tuples forming the itemsets, output by Algorithm 1, are considered as words. Thus, each word is comprised of a node and the order of a random walk in which that node was identified as connected to the node for which the representation is being constructed.
For the purpose of BOW construction, we consider the multiset of (node ID, walk length) tuples, generated by random walks that start at node n. This multiset is viewed as a "*node document*", consisting of individual words—i.e. the tuples contained in the multiset. The number of total features, d, is a parameter of the SGE algorithm. We consider the following variations of this paradigm for transforming each node "document" $\mathfrak{T}_n$ into one feature vector of size d:

- **Binary**. In a binary conversion, the values of the vector represent the presence or absence of a given tuple k-gram (a combination of k tuples; k is a free parameter of SGE) in the set of random walks associated with a given node document. The features, represented by such tuple k-grams, can have values of either 0 or 1.
- **TF**. Here, counts of a given k-gram t in a given node document $\mathfrak{T}_n$ are used as feature values ($TF_{t,\mathfrak{T}_n}$). The values are integers. Note that $TF_{t,\mathfrak{T}_n}$ represents the multiplicity of a given tuple k-gram in the multiset (node document).
- **TF-IDF**. Here, TF-IDF weighting scheme is employed to weight the values of individual features. The obtained values are real numbers. Given

a tuple k-gram t and the transaction database $\mathfrak{T}$, it is computed as:

$$\text{TF-IDF}(t,n) = (1 + \log \text{TF}_{t,\mathfrak{T}_n}) \cdot \log \frac{|N|}{\mathfrak{T}_t},$$

where $TF_{t,\mathfrak{T}_N}$ is the number of t's occurrences in a given document of node n and $\mathfrak{T}_t$ is the overall occurrence of this k-gram in the whole transaction database.

FP-Growth. This well known variant of association rule learning [4] constructs a specialized data structure termed fp-tree, which is used to count combinations of tuples of different lengths. It is more efficient than the well known Apriori algorithm [3, 21].
For the purpuse of FP-growth, the obtained multiset $\mathfrak{T}$ is viewed as a set of *itemsets* - a transaction database. For each itemset, only the set of unique tuples is considered as the input while their multiplicity is ignored. The FP-Growth algorithm next considers such non-redundant transaction databse $\mathfrak{T}$ to identify frequent combinations of tuples, similarly to the TF and TF-IDF schemes described above. The free parameter we consider in this work is *support*, which controls how frequent tuple combinations shall be considered. Similarly to TF and TF-IDF schemes, once the representative tuple combinations are obtained, they are considered as features, whose values are determined based on their presence in a given node document, and are binary (0 = not present, 1 = present). Note that some of these features may correspond to the features generated by TF-IDF, however, in the case of FP-growth, we allow sizes of tuple combinations to be arbitrary, rather than fixed to k. Also unlike the BOW approaches, the dimension of the constructed feature vectors constructed is not fixed but is controlled implicitly by varying the value of the *support* parameter.

SGE formulation The formulation of the whole approach is given in Algorithm 2. Here, first the sampling vector w is constructed. Next, the vector is traversed. The i-th component of vector w represents the number of walks of length i that will be simulated. For each component of w cell, a series of random walks (lines 6-8) is simulated, which produces sequences of nodes that are used to fill a node-level walk container $\mathcal{D}$. Thus, $\mathcal{D}$, once filled, consists of o sets representing individual random walks of length α. The walks are added into a single multiset, prior to being stored into the global transaction structure $\mathfrak{T}$. Once w is traversed, frequent patterns are found (line 11), where the transaction structure $\mathfrak{T}$ comprised of all node-level walks is used as the input. The findPatterns method in line 11 can be any method that takes a transaction database as input, the considered ones are discussed in the following section. The top most frequent d patterns are used as features, and represent the columns (dimensions) of the final representation $\mathcal{M}$. Here, the representNodes method (line 12) fills the values according to the considered weighting scheme (part of r)[4].

[4] Note that this method takes as input random walk samples for *all* nodes.

Algorithm 2: Symbolic graph embedding.

Data: A graph $G = (N, E)$
Parameters : Number of walk samples ν, sampling distribution η, pattern finder r, embedding dimensionality d, starting node n_i
Result: Symbolic node embedding M

$\tau :=$ generateSamplingVector(η, ν);
$\mathfrak{T} \leftarrow$ multiset;
for $o \in \tau$ **do**
  $\alpha \leftarrow o$'s index ; ▷ `Walk length.`
  $\mathcal{D} \leftarrow \{\}$;
  for $k \in [1 \dots o]$ **do**
    $\mathcal{D} \leftarrow \mathcal{D} \cup$ Walk(G, n_i, α) ; ▷ `Sample with Algorithm 1.`
  end
  $\mathfrak{T} \leftarrow \mathfrak{T} \cup \mathcal{D}$; ▷ `Update walk object.`
end
$\mathcal{P} \leftarrow$ findPatterns($\mathfrak{T}$, r) ; ▷ `Find patterns.`
$\mathcal{M} \leftarrow$ representNodes($\mathfrak{T}$, $\mathcal{P}$, r, d) ; ▷ `Represent nodes.`
return $\mathcal{M}$;

4 Computational and spatial complexity

In this section we discuss the computational aspects of the proposed approach. We split this section into two main parts, where we first discuss the complexity of the sampling, followed by the pattern mining part.

The time complexity of the proposed sampling strategy depends on the number of simulated walks and the walk lengths. The complexity of a single walk is linear with respect to the length of the walk. If we define the average walk length as $\bar{l}$, and the number of all samples as ν, the spatial complexity, required to store all walks amounts to $\mathcal{O}(|N| \cdot \nu \cdot \bar{l})$. As the complexity of a single random walk is linear with respect to the length of the walk, the considered sampling's time complexity amounts to $\mathcal{O}(\nu \cdot \bar{l})$ for a single node. The proposed approach is also linear with respect to the number of nodes both in space and time.

The computational complexity of pattern mining varies based on the algorithm used for this step. The considered FP-Growth's complexity is linear with respect to the number of transactions, whereas its spatial complexity is, due to efficient counting employed, similarly efficient and does not explode as for example with the Apriori family of algorithms. The result of the pattern mining step is a $|N| \times d$ matrix, where d is the number of patterns considered as features. Compared to e.g., metapath2vec and other shallow graph embedding methods, which yield a dense matrix, this matrix is *sparse*, and potentially requires orders of magnitude less space for the same d[5]. As storing large dense matrices can be spatially demanding, the proposed sparse feature representation requires less space, especially if high-dimensional embeddings are considered (the black-box

[5] In practice, however, larger dimensions are needed to represent the set of nodes well by using symbolic representations.

methods commonly yield dense representation matrices). The difference arises especially for very large datasets, where dense node representations can become a spatial bottleneck. We observe that $\approx 10\%$ of elements are non-zero, indicating that storing the feature space as a sparse matrix results in smaller time complexity. Worst case spatial complexity of storing the embedding, however, is for both types of methods $\mathcal{O}(|N| \cdot d)$.

5 Empirical evaluation

In this section we present the evaluation setting, where we demonstrate the performance of the proposed Symbolic Graph Embedding approach. We follow closely the evaluation introduced by metapath2vec, where the representation is first obtained, and next used for the classification task, where logistic regression is used as a classifier of choice. We test the performance on a heterogeneous information graph, comprised of authors, papers and venues[6]. The task is to classify venues into one of eight possible topics. The dataset was first used for evaluation of metapath2vec, hence we refer to the original results when comparing with the proposed approach. The considered graph consists of 2,766,148 nodes and 2,503,628 edges, where the 133 venues are to be classified into correct classes. We compare SGE against previously reported performances [7] of DeepWalk [23], LINE [29], PTE [28], metapath2vec and metapath2vec++ [7]. All methods are considered state-of-the-art for black-box node representation learning. The PTE and two variations of metapath2vec can take into account different (typed) paths during sampling.

We tested the following SGE variants. For pattern learning, we varied the TF-IDF, BoW and TF-, as well as the FP-Growth methods. The parameter search space used to obtain the results was as follows. The number of features = [500,1000,1500,2000,3000], considered vectorizers = ["TF-IDF","TF","FP-growth","Binary"], relation order (relevant for TF-based vectorizers—the highest k-gram order considered) = [2, 3, 4], walks of lengths = [2, 3, 5, 10], and number of walk samples = [1000, 10000] were considered. The support parameter of the FP-Growth parameter was varied in the range [3, 5, 8]. The Uniform walk length distribution was used. In addition to the proposed graph sampling (FS), a simple breadth-first search (BFS) that explores neighborhood of order two was also tested. We report the best performing learners' scores based on the type of the vectorizer and the sampling distribution. Ten repetitions of ten-fold, stratified cross validation is used, the resulting micro and macro F1 scores are averaged to obtain the final performance estimate. We report the performance of logistic regression classifier when varying the percentage of training data.

5.1 Results

In this section we discuss in detail the results for the node classification task. The results in Table 1 are presented in terms of micro and macro F1 scores, with

[6] Accessible at `https://ericdongyx.github.io/metapath2vec/m2v.html`

respect to training set percentage. We visualize the performance of the compared representations in Figure 2.

Table 1. Numeric results of the proposed SGE approach compared to the state-of-the-art approaches, presented in terms of micro and macro F1 scores, with respect to training set percentage. Best performing approaches are highlighted in green.

Method / Percentage	10%	20%	30%	40%	50%	60%	70%	80%	90%
Macro-F1									
DeepWalk/node2vec	0.140	0.191	0.280	0.343	0.391	0.442	0.478	0.496	0.446
LINE (1st+2nd)	0.463	0.701	0.847	0.895	0.920	0.931	0.947	0.941	0.947
PTE	0.170	0.654	0.830	0.894	0.921	0.935	0.951	0.953	0.949
metapath2vec	0.525	0.803	0.897	0.940	0.953	0.953	**0.970**	**0.968**	**0.967**
metapath2vec++	0.544	0.805	0.900	**0.947**	**0.958**	**0.956**	0.968	0.953	0.950
SGE (binary + FS)	**0.815**	**0.883**	**0.918**	0.919	0.922	0.937	0.942	0.931	0.950
SGE (TF + FS)	0.716	0.769	0.826	0.853	0.875	0.886	0.906	0.914	0.919
SGE (TF-IDF + FS)	0.364	0.114	0.121	0.03	0.354	0.353	0.363	0.148	0.146
SGE (FP-growth + FS)	0.523	0.684	0.712	0.771	0.785	0.815	0.801	0.816	0.838
SGE (Binary + BFS)	0.396	0.589	0.685	0.710	0.702	0.772	0.792	0.759	0.778
SGE (TF + BFS)	0.054	0.058	0.070	0.087	0.091	0.083	0.094	0.088	0.091
SGE (TF-IDF + BFS)	0.360	0.090	0.113	0.047	0.324	0.321	0.325	0.122	0.122
SGE (FP-growth + BFS)	0.400	0.547	0.586	0.591	0.594	0.646	0.587	0.609	0.553
Micro-F1									
DeepWalk/node2vec	0.214	0.249	0.327	0.379	0.409	0.463	0.498	0.526	0.529
LINE (1st+2nd)	0.517	0.716	0.846	0.895	0.920	0.933	0.950	0.956	0.957
PTE	0.427	0.688	0.837	0.895	0.924	0.935	0.955	0.967	0.957
metapath2vec	0.598	0.833	0.901	0.940	0.952	0.954	**0.973**	**0.982**	**0.986**
metapath2vec++	0.619	0.834	0.903	**0.946**	**0.958**	**0.957**	0.970	0.974	0.979
SGE (Binary + FS)	**0.815**	**0.880**	**0.918**	0.918	0.921	0.935	0.940	0.933	0.964
SGE (TF + FS)	0.718	0.771	0.824	0.850	0.872	0.881	0.900	0.911	0.921
SGE (TF-IDF + FS)	0.477	0.231	0.245	0.138	0.518	0.522	0.528	0.289	0.279
SGE (FP-growth + FS)	0.515	0.655	0.700	0.758	0.775	0.807	0.800	0.815	0.864
SGE (Binary + BFS)	0.388	0.557	0.657	0.692	0.678	0.750	0.785	0.759	0.821
SGE (TF + BFS)	0.148	0.150	0.155	0.168	0.175	0.167	0.172	0.159	0.193
SGE (TF-IDF + BFS)	0.461	0.195	0.226	0.144	0.469	0.467	0.478	0.252	0.250
SGE (FP-growth + BFS)	0.381	0.512	0.553	0.558	0.563	0.619	0.577	0.600	0.607

The first observation is that shallow node embedding methods, e.g., node2vec and LINE, do not perform as well as the best performing SGE variants (Binary with Uniform sampling). Further, we can observe that best performing SGE also outperforms metapath2vec and metapath2vec++, indicating that symbolic representations can (at least for this particular dataset) offer sufficient node description. The best performing SGE variant was the simplest one, with simple binary features obtained via fast sampling. Here, 10,000 walks were sampled and feature matrix of dimension 3000 was considered along with up to three-gram patterns. Finally, we visualized the embeddings by projecting them to 2D using the UMAP algorithm [19]. The resulting visualization, shown in Figure 3, shows that the obtained symbolic node embeddings maintain the class structure of the data.

5.2 Implementation details and reproducibility

In this section we discuss the details of the proposed SGE. The main part of the implementation is Python-based, where Numpy [30] and Scipy [13] libraries were used for efficient processing. The Py3plex library[7] was used to parse the heteroge-

[7] `https://github.com/SkBlaz/Py3plex`

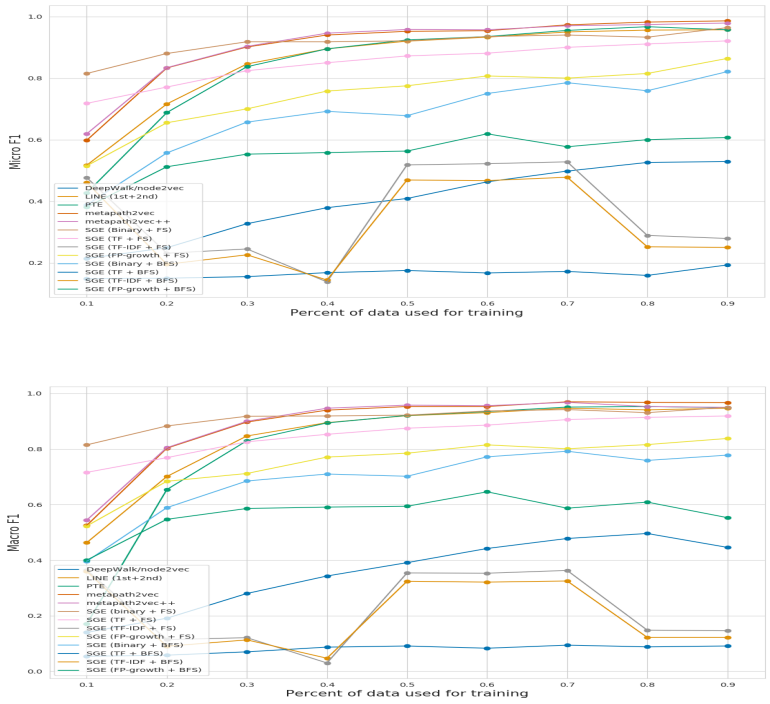


Fig. 2. Micro and macro F1 performance with respect to train percentages.

neous graph used as input [26]. The final graph was returned as a MultiDiGraph object compatible with NetworkX [10]. The TF, TF-IDF and Binary vectorizers implementations from the Scikit-learn library [20] were used. As the main bottleneck we recognized the graph sampling, which we further implemented using the Numba [15] framework for production of compiled code from native Python. After re-implementing the walk sampling part in Numba, we achieved approximately 15x speedup, which was enough to consider up to 10,000 walk samples of the large benchmark graph used in this work[8].

6 Discussion and conclusions

In this work we compared the down-stream learning performance of symbolic features, obtained by sampling a given node's neighborhood, to the performance of black-box learners. Testing the approaches on the venue classification task, we find that Symbolic Graph Embedding offers similar performance on a large, real-world graph comprised of millions of nodes and edges. The proposed method

[8] The code repository is available at `https://github.com/SkBlaz/SGE`

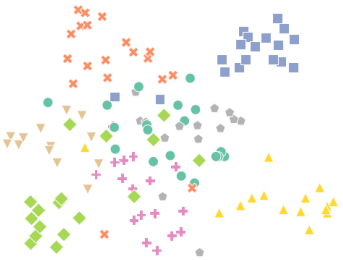


Fig. 3. UMAP projection of the best performing SGE embedding into 2D. Colors represent different types of venues (the class to be predicted). It can be observed that the obtained embeddings maintain the class-dependent structure, even though they were constructed in a completely unsupervised manner. The visualization was obtained using UMAP's default parameters.

outperforms the state-of-the-art shallow embeddings by up to ≈65%, and heterogeneous graph embeddings by up to ≈27% when only small percentages of the representation are used for learning (e.g., 10%). The method performs comparably to metapath2vec and metapath2vec++ when the whole embedding is considered for learning. One of the most apparent results is the well performing Binary + Uniform SGE, which indicates that simply checking the presence of relational features potentially offers enough descriptive power for successful classification. This result indicates that certain graph patterns emerge as important, where their presence or absence in a given node's neighborhood can serve as relevant for classification. The TF-IDF-based SGE variants performed the worst, indicating that more complex weighting schemes are not as applicable as in the other areas of text mining. We believe the proposed methodology could be further compared with RDF2vec and similar triplet embedding methods. The obtained symbolic embeddings were also explored qualitatively, where UMAP projection to 2D was leveraged to inspect whether the SGE symbolic node representations group according to their assigned classes. Such grouping indicates potential quality of the embedding, as venues of similar topics should be clustered together in the latent space.

7 Acknowledgements

We acknowledge the financial support from the Slovenian Research Agency through core research programmes P2-0103 and P6-0411 and project *Semantic Data Mining for Linked Open Data* (financed under the ERC Complementary Scheme, N2-0078). The authors have received funding also from the European Unions Horizon 2020 research and innovation programme under grant agreement No 825153 (EMBEDDIA). The work of the second author was funded by the

Slovenian Research Agency through a young researcher grant (BS). We would finally like to thank to Jan Kralj for his insightful comments on formulation of the proposed framework and mathematical proofreading.

Bibliography

[1] Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB. vol. 1215, pp. 487–499 (1994)
[2] Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence **35**(8), 1798–1828 (2013)
[3] Borgelt, C.: Efficient implementations of apriori and eclat. In: FIMI03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations (2003)
[4] Borgelt, C.: An Implementation of the FP-growth Algorithm. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations. pp. 1–5. ACM (2005)
[5] Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global rdf vector space embeddings. In: International Semantic Web Conference. pp. 190–207. Springer (2017)
[6] Dash, T., Srinivasan, A., Vig, L., Orhobor, O.I., King, R.D.: Large-scale assessment of deep relational machines. In: International Conference on Inductive Logic Programming. pp. 22–37. Springer (2018)
[7] Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 135–144. ACM (2017)
[8] França, M.V., Zaverucha, G., Garcez, A.S.d.: Fast relational learning using bottom clause propositionalization with artificial neural networks. Machine learning **94**(1), 81–104 (2014)
[9] Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864. ACM (2016)
[10] Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy) (1 2008)
[11] Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data mining and knowledge discovery **15**(1), 55–86 (2007)
[12] Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. In: European conference on principles of data mining and knowledge discovery. pp. 13–23. Springer (2000)
[13] Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–), `http://www.scipy.org/`

[14] Kralj, J., Robnik-Šikonja, M., Lavrač, N.: HINMINE: heterogeneous information network mining with information retrieval heuristics. Journal of Intelligent Information Systems **50**(1), 29–61 (Feb 2018)
[15] Lam, S.K., Pitrou, A., Seibert, S.: Numba: A llvm-based python JIT compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. p. 7. ACM (2015)
[16] Lavrač, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood (1994)
[17] Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 631–636. ACM (2006)
[18] Maiya, A.S., Berger-Wolf, T.Y.: Sampling community structure. In: Proceedings of the 19th international conference on World wide web. pp. 701–710. ACM (2010)
[19] McInnes, L., Healy, J., Saul, N., Grossberger, L.: Umap: Uniform manifold approximation and projection. The Journal of Open Source Software **3**(29), 861 (2018)
[20] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of machine learning research **12**(Oct), 2825–2830 (2011)
[21] Perego, R., Orlando, S., Palmerini, P.: Enhancing the apriori algorithm for frequent set counting. In: International Conference on Data Warehousing and Knowledge Discovery. pp. 71–82. Springer (2001)
[22] Perovšek, M., Vavpetič, A., Kranjc, J., Cestnik, B., Lavrač, N.: Wordification: Propositionalization by unfolding relational data into bags of words. Expert Systems with Applications **42**(17-18), 6442–6456 (2015)
[23] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proc. of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
[24] Ristoski, P., Paulheim, H.: Rdf2vec: Rdf graph embeddings for data mining. In: International Semantic Web Conference. pp. 498–514. Springer (2016)
[25] Shi, C., Hu, B., Zhao, W.X., Philip, S.Y.: Heterogeneous information network embedding for recommendation. IEEE Transactions on Knowledge and Data Engineering **31**(2), 357–370 (2018)
[26] Škrlj, B., Kralj, J., Lavrač, N.: Py3plex: a library for scalable multilayer network analysis and visualization. In: International Conference on Complex Networks and their Applications. pp. 757–768. Springer (2018)
[27] Srinivasan, A.: The aleph manual (2001)
[28] Tang, J., Qu, M., Mei, Q.: Pte: Predictive text embedding through large-scale heterogeneous text networks. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1165–1174. ACM (2015)
[29] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web. pp. 1067–1077. International World Wide Web Conferences Steering Committee (2015)

[30] Walt, S.v.d., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Computing in Science & Engineering **13**(2), 22–30 (2011)

[31] Zhang, Y., Jin, R., Zhou, Z.H.: Understanding bag-of-words model: a statistical framework. International Journal of Machine Learning and Cybernetics **1**(1-4), 43–52 (2010)