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.

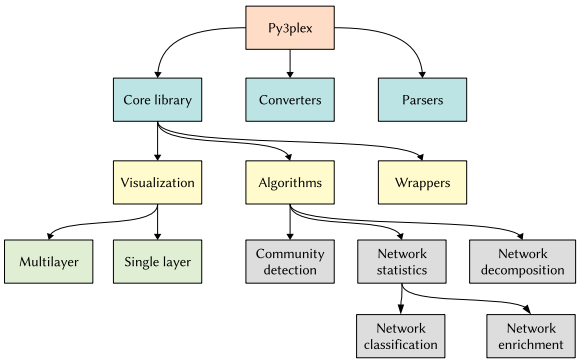


Fig. 1: The Py3plex library architecture. The library is organized in hierarchical manner — a set of core data structures (blue) is used in algorithms for visualization and analysis of multialyer networks.

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

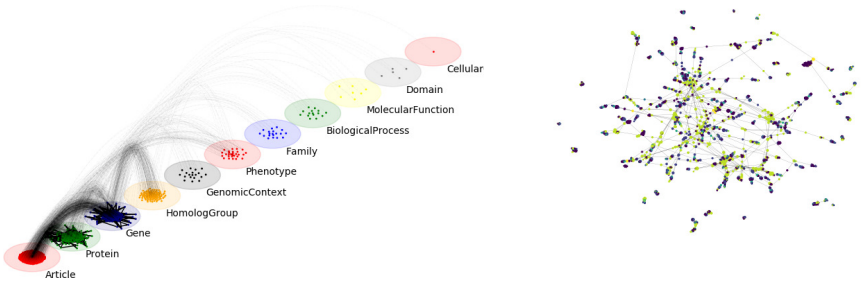
Individual, single-layer networks are derived from the existing NetworkX graph library [10] hence the rich set of functionalities available in NetworkX can be used. Py3plex provides the means to customize majority of network properties, including edge shapes and colors, individual layer layouts, node 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:

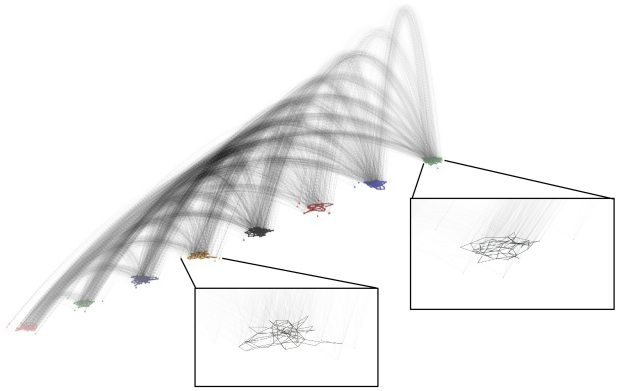
$p \in \{0.05, 0.1, 0.2, 0.3\}$,
$N \in \{5, 10, 20, 50, 80, 100\}$,
$L \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

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.

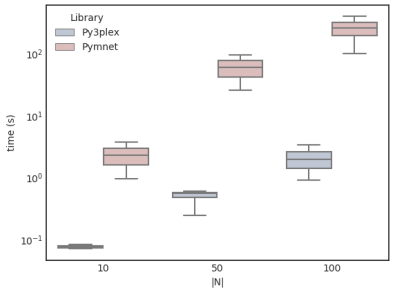
We show the final results of our experiment in the form of box plots, where $|N|$ or $|E|$ is plotted on the x-axis and the time needed is plotted on the y-axis (Figure 3). Networks with more than 100 nodes were not considered for this benchmark, as it took Pymnet more than two hours for visualization. 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

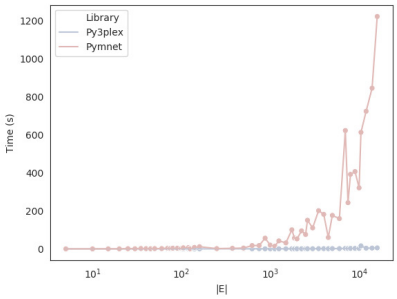
We have developed and implemented Py3plex, a network analysis and visualization library focused on multilayer networks. Apart from most of the functionality offered by the state-of-the-art libraries, Py3plex introduces a novel visualization, suitable for larger



(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)