Applied Network Science

RESEARCH Open Access

Current challenges in multilayer network engineering

Georgios Panayiotou[1*], Matteo Magnani[1] and Bruno Pinaud[2]

*Correspondence:
georgios.panayiotou@it.uu.se

[1] InfoLab, Department of Information Technology, Uppsala University, Uppsala, Sweden
[2] University of Bordeaux, Bordeaux INP, LaBRI, UMR 5800, Bordeaux, France

Abstract

Multilayer networks (MLNs) have become a popular choice to model complex systems. However, current MLN engineering solutions, that is, systems and methods to store, manipulate, and support the analysis of MLNs, are challenged by the size and complexity of contemporary sources of network data. We assess the maturity level of the MLN engineering ecosystem through an analysis of software libraries for MLNs, focusing on supported functionality, operators and their scalability. Based on this analysis, we provide an overview of the current status of the MLN engineering landscape, compile a list of current limitations to be addressed and propose future developments for more effective and broadly applicable MLN engineering solutions.

Keywords: Multilayer networks, Challenges, Engineering, Software, Taxonomy, Multiplex networks

Introduction

Multilayer networks (MLNs) have become increasingly popular across disciplines for representing, manipulating, and analyzing complex systems (Kivela et al. 2014; Boccaletti et al. 2014; Dickison et al. 2016; Bianconi 2022). While special types of MLNs have been used for a long time in social network analysis (Bott 1928; Moreno and Jennings 1934), the range of application areas has recently expanded, including brain (Timme et al. 2014; De Domenico 2017; Vaiana and Muldoon 2020), transportation (Cardillo et al. 2013; Gallotti and Barthelemy 2014; Aleta et al. 2017), ecological (Pilosof et al. 2017; Timóteo et al. 2018; Finn et al. 2019), biomedical (Hammoud and Kramer 2020; Kinsley et al. 2020; Mondal et al. 2020), and online communication networks (Magnani and Rossi 2011; Hristova et al. 2016; Hanteer et al. 2018). This breadth of application highlights the value of research focusing on the design, building, and use of systems for MLNs — what we here call MLN engineering.

Despite this extensive range of applications, recent work has questioned whether the current landscape of MLN engineering is mature enough to address the challenges associated to contemporary sources of data (Finn et al. 2019; Kinsley et al. 2020; Finn 2021). One challenge is that, for complex secondary data,[1] there are multiple possible MLN

[1] Data originally collected for purposes other than the analysis to be performed.

modelling choices. For example, layers can be used to model different parts of social media data depending on the analysis task (Magnani and Rossi 2011; Hanteer et al. 2018; Ustek-Spilda et al. 2021; Santra et al. 2022). This indicates a need for an expressive set of data manipulation operators (i.e. a query language) to interactively explore possible alternative MLNs. Manipulation operators for MLNs are also needed to support interactive visual analysis systems (McGee et al. 2021), and can reduce the amount of ad-hoc scripting, thus decreasing data preprocessing costs (Interdonato et al. 2020). As another example of a challenge, to fully exploit contemporary sources of digital data we must be able to handle increasingly larger networks, such as MLNs representing the population of entire countries (Bokányi et al. 2023; Kazmina et al. 2023), which further increases computational demands.

In this paper we investigate the status of MLN engineering through an analysis of software programs natively supporting MLNs. We first provide a taxonomy of operators to store, manipulate and analyze MLNs, to identify which functionality can be expected in MLN systems and what is currently missing compared with more established data engineering solutions. We then focus on scalability through an experimental study. These analyses combined allow us to draw a picture of current limitations and challenges in MLN engineering, establishing it as a distinct research area in need of further research, providing an overview of the area, and proposing future developments. In addition, on a more practical level, our findings can also help researchers and data scientists choose the MLN software that best fits their needs. For example, we look at which data features, types of networks and operators are supported by different software, making them more or less appropriate for different types of MLN engineering tasks.

Other comparative studies

Despite the broad literature on MLNs, works summarizing available operators for MLNs and comparisons of them are scarce. General overviews on MLNs (Kivela et al. 2014; Boccaletti et al. 2014; Dickison et al. 2016; Bianconi 2022) cover topics such as model, structure, and analytics, but do not look at which operators are practically available and usable in software. Similarly, relevant studies on specific sub-tasks for MLNs, such as preprocessing (Interdonato et al. 2020), conceptual design (Santra et al. 2022) and visualization (McGee et al. 2019, 2021) neither look at available operators, nor perform experimental comparisons. Here, we note that visualization-oriented task taxonomies for MLNs have been introduced in McGee et al. (2019, 2021). Other studies include layer similarity (Brodka et al. 2018; Ghawi and Pfeffer 2022), structural analysis (Giordano et al. 2019) and community detection (Magnani et al. 2021), the latter also looking at algorithm scalability. However, these studies only focus on specific operators.

While a variety of software for MLN analysis are available online, there are few studies comparing them. Only a subset of the libraries and operators covered in this article are considered in Škrlj et al. (2019), and the experimental comparison is limited to the processing time for visualisation tasks. An experimental comparison of database management systems for a multilayer doctor-patient network can be found in Mondal et al. (2020). Software implementations of multilayer networks in the context of biomedicine are discussed in Hammoud and Kramer (2020), although there is no comparison between the implementations. Finally, a matching of libraries with different tasks in

animal behaviour research is done in Finn et al. (2019). This comparison is specific to those tasks, and does not contain an experimental part.

Research design

The current landscape of MLN engineering is rich and rapidly evolving, with new software and algorithms introduced frequently, so it is important to clearly define the scope of this work and provide a way to evaluate which new software should be included in future versions of this study. A consequence of performing a study based on software (compared, for example, with a literature review, which would not highlight what functionality is actually available in MLN systems) is that our taxonomy only includes operators found in the selected libraries.

Furthermore, our study is not designed to draw conclusions about the effect of specific design choices, because our list of selected MLN software consists of a variety of programming languages, data structures and related but not identical operators. For example, we cannot assess the suitability of specific data structures for specific tasks without re-implementing the data structures under comparable conditions, since execution time is the result of a combination of this and all other design choices (Kriegel et al. 2017).

The rest of the paper is organized as follows: In Sect. "Multilayer network analysis software", we discuss our selection of representative MLN software. We provide a taxonomy of common operators in Sect. "Models and operators", and in Sect. "Scalability", we experimentally compare the software on selected tasks. We discuss the current limitations in MLN engineering through the findings of the present study in Sect. "Discussion", along with areas for future explorations in the field. Finally, we conclude in Sect. "Conclusion".

Multilayer network analysis software

Given the popularity of MLNs across disciplines, the question of which software should be included in our study is complex. First, different definitions of MLN exist, and there are also network meta-models not called MLNs but still providing similar data representation constructs and operators. Therefore, in Sect. "Multilayer networks", we provide a definition of the meta-model and terminology used in this paper, which is largely based on the work by Kivela et al. (2014). We also acknowledge some alternatives, which is useful because part of our comparison can be applied or extended to related network meta-models. Then, in Sect. "Criteria for inclusion", we provide the criteria we have used to select the software included in this study. In Sect. "Software included", we list the software we used along with a short overview of them. Finally, in Sect. "Other relevant software" we list other software relevant to MLN analysis and why we decided not to include them. Overall, we give enough details for any practitioner or researcher on MLNs interested in reproducing or extending our experiment with, for instance, new software or criteria we have not considered.

Multilayer networks

The following is a summarised version of the definition of an MLN by Kivelä et al. (2014), that we use as a reference throughout the paper:

Definition 1 *(Multilayer network)* Let $L_1, \ldots, L_d$ be sets of elementary layers called aspects. A *multilayer network* (MLN) is a quadruplet $M = (V_M, E_M, V, \mathbf{L})$, where V is a set of vertices, $\mathbf{L} = L_1 \times \cdots \times L_d, V_M \subseteq V \times \mathbf{L}$ and $E_M \subseteq V_M \times V_M$.

Remaining consistent with the original model, a *layer* is simply an element of $\mathbf{L}$. We refer to elements of V_M as *nodes* – not to be confused with *vertices*, the elements of V. See Fig. 1 for an example of such a network.

Special types of MLNs have been used for a long time in social network analysis, including networks with multiple edges between a common set of actors (multi-relational), with multi-valued edges (multiplex), with different types of nodes (two-mode, multi-mode), and with nodes representing both individuals and groups or organisations (multi-level) (Borgatti et al. 2009).

In addition, meta-models resembling the above definition have been previously studied by the data engineering community. A popular example is the heterogeneous information network (Sun et al. 2022; Shi et al. 2017), which is a special case of attributed MLN. Other similar models include the multidimensional graph online analytical processing (OLAP) cubes (Chen et al. 2009) and the recently proposed multilayer graphs, an extension for property graphs that can be used to model complex layered relationships (Angles et al. 2022). While we do not expand on the aforementioned models in this study, all of the previous are based on complex graph types covered by the MLN definition found in Kivela et al. (2014).

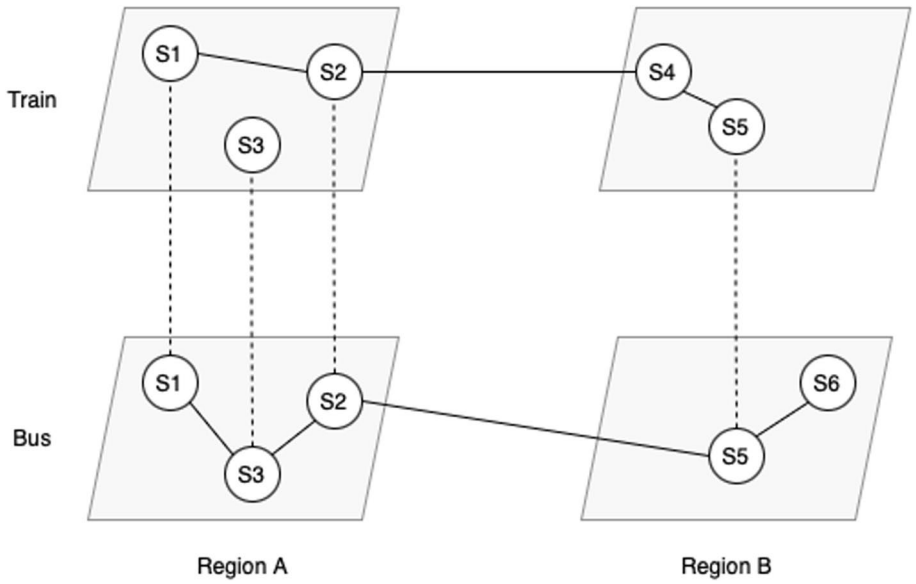

Fig. 1 Example of an MLN representing a public transportation network in two regions. The nodes represent stations, and an edge between two nodes represents a route between them. Following Def. 1, this network has two dimensions $L_1 = \{\text{RegionA}, \text{RegionB}\}$ and $L_2 = \{\text{Train}, \text{Bus}\}$, representing the regions and transport methods respectively, and four layers: $\mathrm{L} = \{(\text{RegionA}, \text{Train}), (\text{RegionA}, \text{Bus}), (\text{RegionB}, \text{Train}), (\text{RegionB}, \text{Bus})\}$. The six vertices in this network, $V = \{\text{S1}, \text{S2}, \text{S3}, \text{S4}, \text{S5}, \text{S6}\}$, do not all participate in all layers. Notice the coupling edges (in dotted lines) which can be used to signify that the same vertex (station) is present in both layers (transportation media)

Table 1 Summary of included MLN software

Library	Version	Prog. language	MLN implementation	Licence	Ref.
MLG.jl	1.1.4	Julia	Graph objects (Graphs.jl)	MIT	(Moroni and Monticone 2023)
Multinet	4.1.2	R	ML-cube objects	Apache 2.0	(Magnani et al. 2021)
MuxViz	3.1	R	Supra-adjacency matrix	GPL 3.0	(De Domenico et al. 2015)
Pymnet	0.1	Python	Supra-adjacency lists	GPL 3.0	(Nurmi et al. 2024)
Py3plex	0.95a	Python	Graph objects (NetworkX)	BSD 3-clause	(Škrlj et al. 2019)

Criteria for inclusion

Software included in this study fulfill the following criteria about their support of MLNs:

C1 Explicit support: the documentation has to explicitly mention the support for MLN and the concept of layer.

C2 Native support: an MLN-native object should be supported, that is, the software should expose an MLN data-structure through its API.

C3 General-purpose: a broad set of functions allowing different types of processing, not designed for a specific application or domain has to be provided.

C4 API-based: a documented API allowing execution from external code/scripts has to be provided.

C5 Publicly available: the source code is available and downloadable under an open source license from a website or a public repository.

C6 Actively maintained: the last release or update has to be within the last year at the time of our experiment (Autumn 2023).

Our criteria reflect desired software component functionality characteristics: suitability (providing an appropriate set of functions for MLN tasks, C1–C3), interoperability (able to interact with other systems, C4–C5), and accuracy (providing correct results, C6) (Carvallo Vega et al. 2007). Other studies have also considered criteria for software popularity, for example by means of the number of downloads (Hoe-Lian et al. 2006). However, we decided not to use it, as the relevant statistics are often not available.

Software included

Below, we provide information about the libraries featured in this study. A summary can be found in Table 1.

MultilayerGraphs.jl (Moroni and Monticone 2023) (hereinafter MLG.jl) is a package for the Julia programming language, extending Julia's graph analysis package with MLN capabilities. It offers functions for manipulation and analysis of multiple types of MLNs. The MLN representation is based on graph objects from the Graphs.jl library.

Multinet (Magnani et al. 2021) is a C++ library, also available in Python and R.[2] It provides features for manipulating, visualizing and mining MLNs, along with methods for community detection and evaluation, plus layer transformation and comparison. It uses

[2] We use the R version of the library in our experiments.

a cube-based MLN representation inspired by the data warehouse multi-dimensional model.

MuxViz (De Domenico et al. 2015) is an R library for MLN analysis. It represents MLNs using supra-adjacency matrices. MuxViz supports manipulation and visualization of MLNs, centrality and structural reducibility analysis, discovery of meso-structures such as communities, components and motifs, as well as analytics for dynamical processes.

Pymnet (Nurmi et al. 2024) is a Python MLN analysis library. It uses a supra-adjacency list representation for MLNs to address scaling needs for larger graphs. It provides functions for manipulating and visualizing MLNs, computing multiple variants of MLN-specific clustering coefficient metrics, as well as testing for auto- and iso-morphisms.

Py3plex (Škrlj et al. 2019) is a Python library offering algorithms for manipulating, analysing and visualizing MLNs, as well as node classification and network embedding methods. It uses lists of NetworkX graph objects to represent MLNs.

Other relevant software

We do not consider software without explicit support for MLNs (i.e. no support for layers), such as the graph analysis libraries graph-tool (Peixoto 2014) and Gephi (Bastian et al. 2009) and libraries where transformation of an MLN to another data structure is required, such as Tulip (Auber et al. 2017), netmem (Espinosa-Rada 2023), DeepGraph (Traxl et al. 2016) and MultiAspectGraphs (Wehmuth et al. 2016).

We also exclude software only providing limited support for general MLNs, e.g. mully (Hammoud and Kramer 2018), EMLN (Frydman et al. 2023), Raphtory (Steer et al. 2023), HuMMuS (Trimbour et al. 2023), POPNET (Bokanyi et al. 2022). This also includes implementations of single algorithms for MLN analysis provided as complementary material for papers, and software programs without a documented and callable interface-API, such as (Coscia 2022; Ostoic 2020; Xia et al. 2015; Vijayaraghavan et al. 2015; Robitaille et al. 2021; Jeub et al. 2019; Galimberti et al. 2020; Gibson and Mucha 2022; Berlingerio et al. 2011; Szárnyas et al. 2016; Interdonato et al. 2017; Perna et al. 2018; Zitnik and Leskovec 2017). While such software can be used in applied studies, they are typically not designed to provide the additional functionality that can be expected in a full-fledged MLN analysis library; namely, an API exposing functions for input and output, manipulation and analysis of the network. Without an API, it is also difficult to experimentally test the software.

Finally, we exclude software for MLNs where the source code is not publicly available or not actively maintained. Examples include multinetX (Kouvaris et al. 2015), Detangler (Renoust et al. 2015), Orion (Heer and Perer 2014) and VERTIGo (Cuenca et al. 2022).

Models and operators

While the literature often refers to the mathematical definition of MLN by Kivela et al. (2014), when we look at MLNs from an engineering perspective we have to ask whether (1) a system uses the same variation of the mathematical definition, and (2) how the definition adopted by the system is implemented. This is a common way of looking at data management systems: for example, while relations (as mathematical objects) in the relational model are often defined as sets, relational database systems are typically based on

bags, and depending on the system a relation can be implemented by row or by column, with this choice impacting the efficiency of different operations.

In this section we first focus on variations of the MLN meta-model used in the included software, both with respect to the supported features (Sect. MLN models), and their implementations (Sect. MLN implementations). Then we look at the operators defined on these MLN models, providing a taxonomy in Sect. MLN operators.

MLN models

In Table 2, we indicate which features of the MLN meta-model are supported by the selected software, as exposed by their API and documentation. In general, we can see that the selected software supports the most distinguishing features of the MLN meta-model: the concept of layer and the existence of edges connecting nodes between different layers (interlayer edges).

However, some features are only supported by some of the libraries. First, most libraries only support one aspect. Indeed, this is sufficient to represent the most common types of MLNs, in particular multiplex networks, where each layer corresponds to a specific type of edge, and multipartite and multilevel networks, where each layer corresponds to a type of vertex. At the same time, a feature of MLNs is that they do not only generalise common network models, but can also combine them into more complex ones, for example allowing the definition of longitudinal multiplex networks (Santana et al. 2017; Matter et al. 2023). Without multiple aspects, the representation capabilities of MLNs are limited.

Second, attributes on nodes, edges and layers are not always supported. It should be noted that attributes are not part of the most commonly cited definition of MLNs (Kivela et al. 2014). This is not surprising, considering attributes can be handled outside the library if vertex, node, and edge identifiers are accessible and stable. At the same time, attributes are not only important to represent complex contemporary data (e.g. text in social media networks, or income in population-scale networks extracted from statistical registers). Once attributes are part of the model, it also becomes possible to define manipulation operators that transform attribute values (Brehmer and Munzner 2013), for example transforming a temporal attribute into layers representing time intervals.

It should also be noted that the ability to represent features of the MLN meta-model does not imply that these features are supported by the operators. For example, weights can be represented by all libraries, but are typically meaningful only if weighted versions

Table 2 Supported MLN features in included software

Feature	MLG.jl	multinet	MuxViz	Pymnet	Py3plex
Layers	✓	✓	✓	✓	✓
Interlayer edges	✓	✓	✓	✓	✓
Aspects				✓	
Directed edges	✓	✓	✓	✓	✓
Weights	✓	✓	✓	✓	✓
Generic attributes	✓	✓			

of the analysis algorithms are also provided, e.g. weighted clustering and shortest path algorithms. Similarly, the fact that interlayer edges can be specified in a library is not particularly useful if there are no (or few) operators that can process them. Finally, directionality is in general supported for simple operators (such as finding out-neighbors, or computing out-degree), but only partially supported by more sophisticated algorithms, e.g. for community detection where only one of the considered algorithms supports this feature.

MLN implementations

We observe three main approaches to implement the MLN meta-model, as summarised in Table 1. One approach extends two common representations used for simple graphs: adjacency matrices and adjacency lists. Supra-adjacency matrices, used by MuxViz, extend the concept of an adjacency matrix for MLNs. In a supra-adjacency matrix, each row and column represents a node (an element of V_M), and each value in the matrix represents the weight of an edge between two nodes. We note that it is not trivial to add and remove nodes from a supra-adjacency matrix, so this implementation is better suited to support analysis than data manipulation. Supra-adjacency lists, used in Pymnet, extend the adjacency list format for MLNs. This is implemented as a dictionary, where each node of M is a key, and its value is a dictionary containing information about its adjacent edges in E_M, while for multiplex networks a list of dictionaries is used. It is designed with large networks in mind.

A second type of MLN implementation re-uses graph objects provided by popular network analysis libraries. This is the case for Py3plex, representing the various layers as NetworkX objects. Similarly, MLG.jl uses the native graph object of the Graphs.jl library to represent layers. Systems adopting this strategy have to then represent the mapping between nodes in different layers, and inter-layer edges.

As a third option, Multinet implements a custom data structure inspired by the multidimensional model used in data warehouses (data cube). A multilayer vertex cube organises its vertices into a number of dimensions (or aspects, using the terminology of Kivela et al. (2014)), and each cell of the cube corresponds to a layer. Similarly, a multilayer edge cube connects two vertex cubes, and its edges can themselves be organised into multiple dimensions. However, currently this library only allows to use one dimension in its R and Python APIs.

MLN operators

The MLN operators provided by the selected libraries are summarized in Table 3, where we have organised them into categories and sub-categories. Defining a taxonomy of MLN operators has several potential benefits: from practical guides for developers to identify missing areas in specific software, to standardisation. In particular, having common operators allows code re-use and enables the definition of benchmarks.

For brevity and clarity, a single row in the table may indicate multiple variants of operators with a similar function, e.g. different operators computing different global

Table 3 Summary of available operators in MLN software

		Operator	MLG.jl	Multinet	MuxViz	Pymnet	Py3plex
Data	I/O	Read network from file		✓	✓	✓	✓
		Write network to file		✓		✓	✓
	Gen	Layer-by-layer generators	✓	✓	✓	✓	✓
		Multilayer model generators		+	+	✓	✓
	Def	Add/remove layer	✓	✓		✓	
		Add/remove attribute	✓	✓			
Manipulation	Basic	Add/remove node	✓	✓		✓	✓
		Add/remove edge	✓	✓	‡	✓	✓
		Set/get attribute values	✓	✓			
	Layer	Layer aggregation		+	✓	✓	✓
		Layer projection		✓			
		Layer selection/subgraph	✓			✓	✓
	Retrieval	(Excl.) neighbours	✓	✓			✓
		Path/distance		✓	✓		
		Components			✓		✓
		Triangles/motifs			+	✓	✓
		Isomorphisms				✓	✓
Analysis	Structural measures	Node degree	✓	✓	✓	✓	✓
		Node centrality	✓	✓	+		✓
		Layer similarity/entanglement		+	✓		✓
		Network density			✓	✓	
		Network reducibility			✓		
		Network entropy	✓		✓		
		Network modularity	✓	✓			✓
		Local clustering coefficient			✓	+	
		Global clust. coeff./transitivity	✓		✓	+	
	Learning	Clustering/community detection		+	✓		+
		Community evaluation metrics		+			✓
		Node classification					✓
		Network embedding					✓
	Dyn	Get supratransition matrix			✓		
		Get coverage evolution			✓		
	Vis	Layer-by-layer layout		✓	✓	✓	✓
		Circular/disc layout		✓	✓	✓	✓
		Force-directed layout		✓	✓	✓	✓

We note (✓) if a feature is available in the corresponding library. A (+) indicates availability of multiple variants of the same feature in the software, and (‡) signifies the availability of a feature by directly manipulating the library's representation for MLNs

clustering coefficient metrics. We only list MLN variants of operators explicitly implemented in the library and found in the library's API documentation.

The focus of this paper is on MLNs, so Table 3 only lists operators taking a whole MLN as input, or producing MLNs as output. Some of the selected libraries also include

operators to independently analyse single layers, or provide functions to access the individual layers (or combinations of layers) using general graph libraries such as igraph or NetworkX. These operators are not reviewed here.

Obtaining and storing MLN data The *Input/Output (I/O)* category includes functions to read and save an MLN from and to a file. It is interesting to see that not all libraries provide output functions, indicating that their typical usage does not involve the generation of new MLNs from existing ones: even if new networks are temporarily created, their creation is intended as a step of the analysis process, and not as a self-contained task generating new data.

We also note that for the libraries providing read or write functions, there is no unique file format. This is a limitation that also affected early graph processing systems, which was partly addressed through the definition of GraphML (Brandes et al. 2002). Generally, the libraries support the multilayer edgelist input format, which however offers limited support for node, edge, and layer metadata. It should be also noted that the libraries use slightly different variants of the edgelist. Namely, Py3plex and MuxViz expect text files with layer and node labels to complement the edgelist, while Multinet's format can also include header and attribute data. Also, while GraphML is used by Pymnet and Multinet as possible formats for input and output respectively, GraphML only allows to save attributed graphs where the attributes are implicitly used to represent the layers.

MLN data can also be obtained using synthetic network generators (see category *Network generation (Gen)*), which is common practice for monoplex networks (i.e. single-layer graphs) using a variety of models such as Erdős-Rényi's random graphs, Barabási-Albert's preferential attachment, Watts-Strogatz's small world etc. The availability of generators suffers from the additional complexity of MLNs, where one has to control both structural features of individual layers (e.g. the degree distribution on each layer) and across layers (e.g. the proportion of common edges or the correlation of node degrees) (Feyer et al. 2023; Magnani and Rossi 2013). While almost all libraries allow to generate a MLN by randomly generating layers one-by-one, there is limited support and no common way to generate a MLN with structurally dependent layers.

The *Data definition (Def)* category includes operators to define data structures, such as creating aspects and layers. The capability to create the supported data structures is, of course, internally available, as it is used for example when an MLN is read from a file. Table 3 only shows which of these data definition capabilities are also exposed through the API, i.e. which ones can be interactively executed by the users of the library.

Comparing this category with the functionality available for other types of data, for example in the relational data meta-model, we note the absence of operators to define efficient data access structures (e.g. indexes) and to impose constraints. The lack of control over constraints is particularly notable. Each special type of MLN is characterised by some data objects that are not allowed; for example, a multiplex network does not allow interlayer edges, and a bipartite network represented as a two-layer MLN does not allow node overlapping. However, there is no direct way to enforce these or other constraints in the examined libraries, and one has to trust that the data satisfies them.

Data manipulation The next three sub-categories in Table 3 refer to basic capabilities typically required by a data manipulation language. *Basic Manipulation* includes operators to update individual elements, e.g. to add a node or an edge to

a layer. *Layer Manipulation* includes operators transforming a layer based on its topology. Examples include layer aggregation, projection, and filtering (for example through selecting a subset of nodes from one or more layers). *Retrieval* lists operators where the objective is to find subgraphs over multiple layers. Examples include finding a node's neighbourhood, whose result depends on the considered layers, paths between nodes which can cross multiple layers, etc.

It is interesting to see how the support for the operators that would provide the building blocks for a data manipulation language is sparse, in some cases even missing explicit API functions for basic operations such as adding a node. Notice that the table indicates operators provided by the API: the absence of an operator does not mean that the operation cannot be performed. For example, a node can be added to a network without an API function to add a node, either by adding it to the input file or by directly modifying the underlying data structure storing the MLN, e.g. by adding columns and rows to the supra-adjacency matrix. However, this suggests that the library is not designed to perform these manipulations.

It is also worth noticing how layer manipulation is only defined based on the layers' structure. There is currently no function to manipulate layers based on edge or node attributes, which points once again at a general limited support for attributes (McGee et al. 2019, 2021). Even when we only focus on structure, we notice how the layer projection operator, providing a basic and common way to process interlayer edges, is only offered by one of the examined libraries.

The *Retrieval* category also suffers from the additional complexity of MLNs if compared with simple networks. Even simple patterns, such as triangles, may correspond to multiple different structures in MLNs where nodes and edges can appear in the same or different layers, leading to a larger number of possible patterns to look for and questions about what different versions of the same pattern mean when found in an MLN.

Data analysis From the previous discussion, it is clear how the selected libraries are more focused on analysis than data storage and manipulation, although such capabilities are also provided. In fact, a variety of operators to study the structure and dynamics of MLNs are available. Under the category *Structural measures* we list operators quantifying different properties of MLNs, such as the number of neighbours of a vertex across a set of layers, or the proportion of common edges inside two or more layers. As for other categories, there is limited overlapping of operators, with MLN versions of functions that are very common for simple networks, such as modularity or clustering coefficients, only supported by some of the libraries.

We can also see some support for machine learning operators, in particular unsupervised and supervised learning (clustering and classification) and representation learning (embedding). Clustering is the only learning task currently supported by multiple libraries, but only two provide more than one method from the many that exist (Magnani et al. 2021).

Capabilities for visual analysis are limited in all cases: while almost all libraries provide ways to visualise the MLN, there is no operator to interact with the visual representation, so that most visual analysis tasks are not currently supported (McGee et al. 2019, 2021).

Scalability

From a data engineering perspective, benchmarks are fundamental tools to improve system performance, for example to compare alternative solutions for the same problem and identify bottlenecks to optimize. However, we are not aware of existing systematic and comparative evaluations of MLN systems. In fact, Sect. Models and operators suggests that it is currently challenging to perform a fair experimental comparison of the selected software. First, there is limited overlap of operators. Second, even when two libraries are marked as providing the same type of operator in Table 3, specific definitions can vary between software. For example, while all libraries can generate MLNs, the way in which relations between different layers are enforced (e.g. controlling the amount of edge overlap, or the correlation of degree centrality) can be very different, if present at all (Brodka et al. 2018).

As a first step towards a benchmark for MLNs, we set up three basic manipulation tasks that we can test on most of the systems considered in this study: loading an MLN, interactively updating it, and aggregating layers. While these tasks are relevant for many MLN studies, execution times are seldomly reported, leaving an open question about the size of MLNs that we are currently able to handle using specialised software.

How long does it take to load an MLN from file, and how large are the networks that can be loaded? A direct comparison of loading times cannot be used alone to draw conclusions about the ability to process large networks, because different data structures are used inside different libraries, and a longer reading time may be caused by the creation of indexing functions that would later speed up other operators. However, this analysis may identify different time complexities (e.g. linear vs super-linear), very large differences in loading time (which can then be matched to execution times for other operators to assess whether loading time is correlated to other execution times), and upper bounds in the size of MLNs that can be currently processed using a personal computer.

How fast can a network be updated? While the literature on MLNs is mostly focused on analysis, the ability to efficiently update a network is useful in many contexts. For example, we update network data while generating it, we may update network data when simulating dynamic processes, and we update networks (or network views) while interactively exploring them (e.g. filtering layers and nodes to zoom into a subset of the data).

How scalable is the aggregation of layers? Aggregating (also known as flattening or merging) layers, that is, creating a single layer by combining edges from multiple layers, is a fundamental manipulation function for MLNs where the initial set of layers does not correspond to the one needed for exploration or analysis.

In Sect. Datasets we present the datasets used in the experiments. Section Settings lists the experiment settings and assumptions. In Sect. Results we provide the results of the experiments.

Datasets

We evaluate the performance of the aforementioned operators for previously studied MLN datasets. Table 4 presents a summary of all datasets used. Specifically, we select five real networks representing: (a) employees at a computer science department at a university in Denmark (Rossi and Magnani 2015) (cs-aarhus), (b) the transportation

Table 4 Summary of MLN data used

Data	#Layers	#Vertices	#Edges	Ref.
Synthetic	2-10000	1000-10M	≈32000-316M	-
cs-aarhus	5	61	620	(Rossi and Magnani 2015)
London-transport	3	369	503	(De Domenico et al. 2014)
Euair-transport	37	450	3588	(Cardillo et al. 2013)
Friendfeed-twitter	2	155804	13.65M	(Magnani et al. 2013)
Friendfeed	3	510896	20.33M	(Celli et al. 2010)

system in London (De Domenico et al. 2014) (london-transport), (c) European airline flight routes (Cardillo et al. 2013) (euair-transport), (d) interactions between users on both the FriendFeed and Twitter platforms (Magnani et al. 2013) (friendfeed-twitter) and, (e) interactions between FriendFeed-only users (Celli et al. 2010) (friendfeed). These are selected in order to evaluate performance and scalability for a variety of network sizes.

We also construct synthetic networks of various sizes to assess the operators in more detail. We generate random node-aligned, single aspect Erdős-Rényi MLNs. We convert the generated networks to all libraries' native input file formats where necessary. To test for various parameters that might affect scalability (that is, the computation time of the operators for increasing network size), we construct sets of synthetic data with increasing number of vertices and layers. To also assess performance with respect to the network density increasing, we generate two sets of synthetic networks, where the average node degree increases with the number of nodes in the network. Specifically, we generate networks for the following settings, where $\langle k \rangle$ denotes the average node degree:

(a) $|V| \in [1000,10\ \mathrm{M}], |\mathbf{L}| = 2, \langle k \rangle \approx 4$
(b) $|V| \in [1000,100\mathrm{K}], |\mathbf{L}| = 2, \langle k \rangle \approx \sqrt{|V_M|}$
(c) $|V| = 1000, |\mathbf{L}| \in [5,10000], \langle k \rangle \approx 4$
(d) $|V| = 1000, |\mathbf{L}| \in [5,10000], \langle k \rangle \approx \sqrt{|V_M|}$

Settings

To assess the performance of tasks such as layer aggregation, a network needs to already be present into memory. Therefore, we have to first load a network in memory before evaluating the manipulation operator. Since the featured software supports various MLN input formats, we need to convert all datasets to the file format accepted by the software. In order to maintain consistency, we always assume that the network is undirected and contains no other attributes or metadata. If the software does not include a network loading operator, as is the case for MLG.jl, we do not consider the rest of its operator runtimes in the comparison. We measure the execution time of each operator individually to ensure that the performance is not affected by any additional calculations or caching by the library.

Finally, we conduct the experiments on a desktop-like environment, which we consider to be the most typical processing environment for MLNs. Specifically, we use a virtual machine running Ubuntu 22.04, with 8 cores @2.1 GHz and 32GB RAM. We halt the execution with a timeout of 30 min. If a network cannot be loaded within the time allotted to each experiment, the following task is also considered to have timed out. These cases indicate that the scalability of the software is an issue, and more computational resources would be needed.

All experiments are repeated four times, and we provide average execution times and standard deviations.

Results

In this section we report the execution times for our three experiments. Note that where runtime information is not provided for a library, the task has not terminated successfully. This signifies that either the timeout per task (30 min) had been reached, or that the process was killed due to a memory overflow.

Network loading from file

As we can observe from Table 5, both large social network datasets friendfeed-twitter and friendfeed become increasingly difficult for the libraries to process within a reasonable time; Multinet is not able to process the friendfeed network before the timeout, probably due to the additional indexing needed for its data structure. We also note

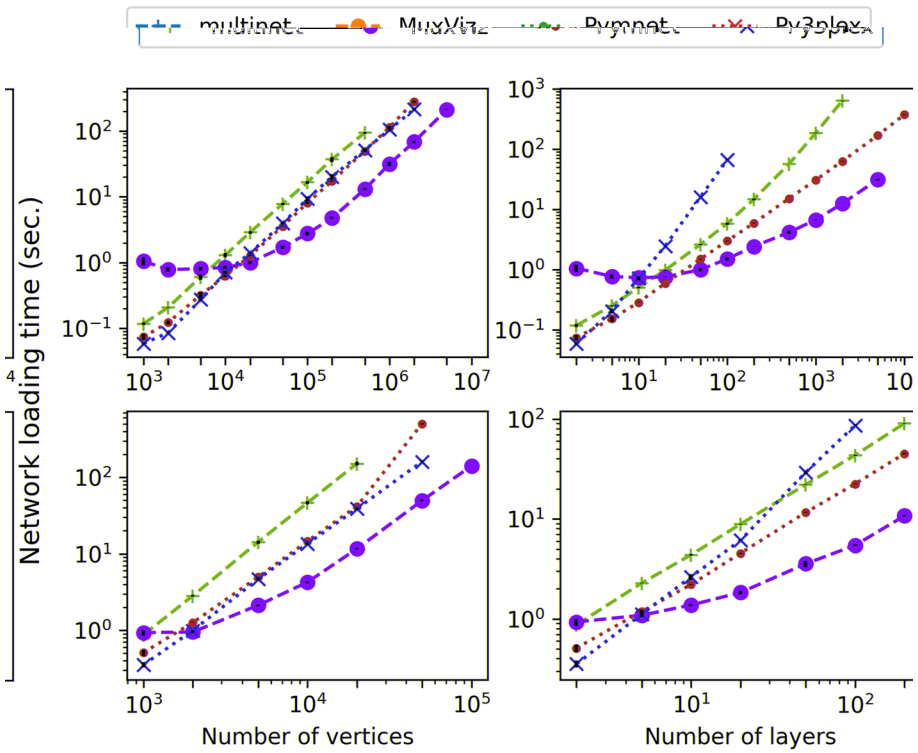

Fig. 2 Network loading time for synthetic data: **a** increasing number of vertices, $|\mathbf{L}| = 2, \langle k \rangle \approx 4$ (top left), **b** increasing number of vertices, $|\mathbf{L}| = 2, \langle k \rangle \approx \sqrt{|V_M|}$ (bottom left), **c** increasing number of layers, $|V| = 1000$, $\langle k \rangle \approx 4$ (top right), and **d** increasing number of layers, $|V| = 1000, \langle k \rangle \approx \sqrt{|V_M|}$ (bottom right). The error bars in black indicate standard deviation over four runs

Table 5 Network loading operator performance

Dataset	Multinet	MuxViz	Pymnet	Py3plex
cs-aarhus	0.0 (0.0)	1.1 (0.3)	0.0 (0.0)	0.0 (0.0)
London-transport	0.0 (0.0)	0.7 (0.1)	0.0 (0.0)	0.0 (0.0)
Euair-transport	0.0 (0.0)	0.7 (0.1)	0.0 (0.0)	2.2 (0.1)
Friendfeed-twitter	418.7 (5.1)	30.7 (1.0)	390.1 (21.6)	106.8 (4.0)
Friendfeed	–	52.1 (1.3)	1044.2 (48.5)	175.6 (2.3)

Rounded values in seconds. We report the average performance and standard deviation (in parentheses) over four runs. A dash indicates that the task did not complete within the timeout

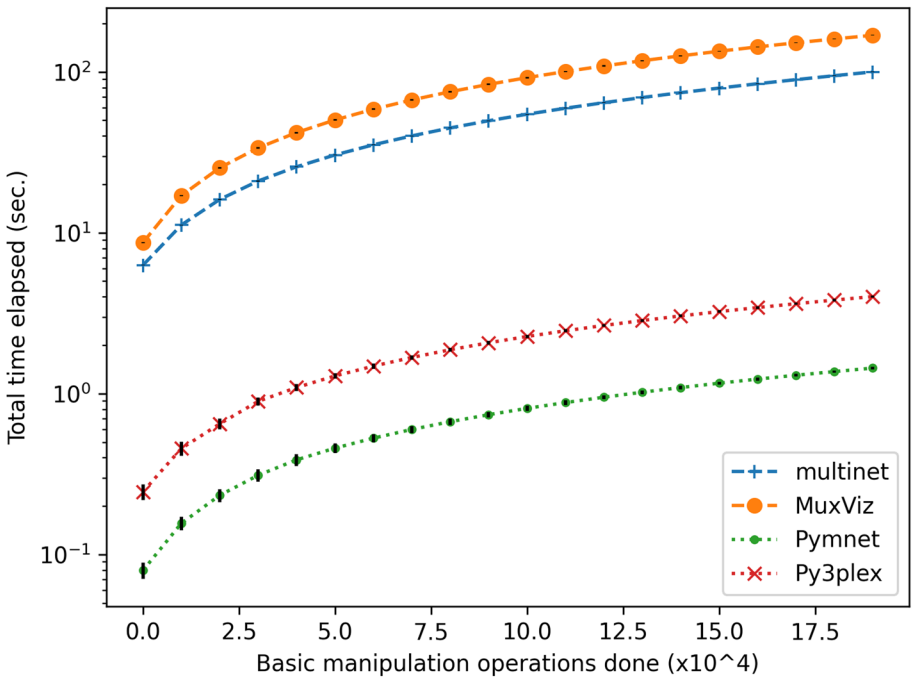

Fig. 3 Total time elapsed in seconds after interactively adding and removing edges from a randomly generated multilayer network with $|V| = 1000, |E_M| = 10000, |\mathbf{L}| = 10$. The error bars in black indicate standard deviation over four runs

the stark performance differences between smaller and larger networks in Table 5; this is directly related to the size of the network. This trend is confirmed when loading large synthetic data. In Fig. 2, we can see most software time out for networks where $|V| > 2$M; the exception is MuxViz, which is able to process a two-layer network with a size up to $|V| = 5$M, but faces memory issues for the largest synthetic dataset.

While adjacency matrix representations perform better for this task, they do not scale well in memory as the network size increases in nodes. However, none of the other parameters examined (i.e. increasing network density, number of layers) directly affects scalability; the only notable exception is the performance of Py3plex for a large number of layers, as we can observe both in Fig. 2 and Table 5 for the euair-transport network. This can be related to the design choice of representing individual layers as separate networks.

Network interactive update

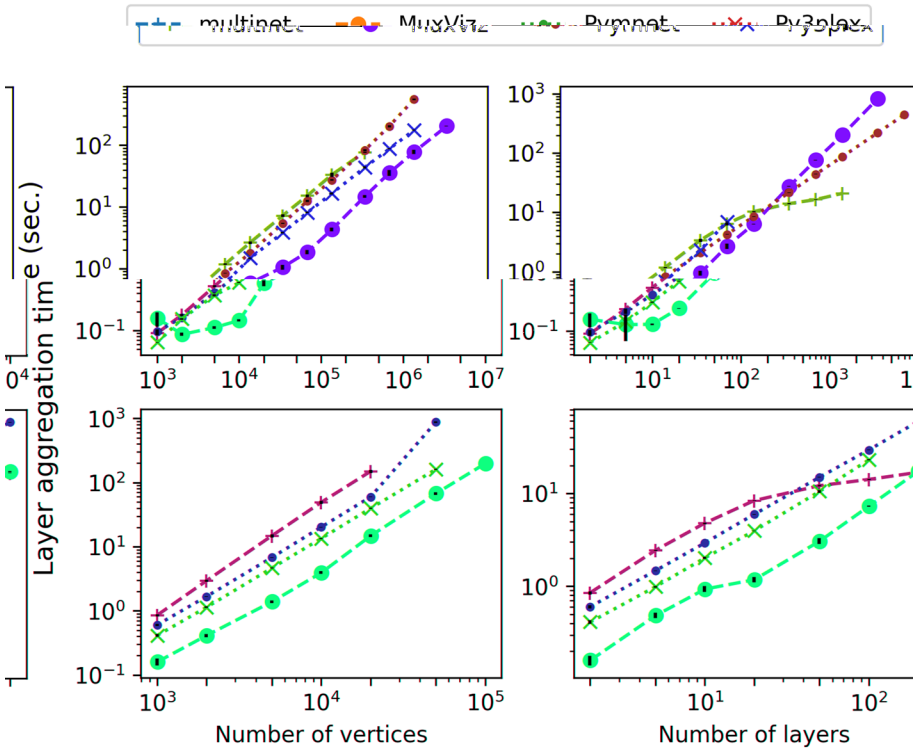

Fig. 4 Layer aggregation time in seconds for synthetic data: **a** increasing number of vertices, $|\mathbf{L}| = 2, \langle k \rangle \approx 4$ (top left), **b** increasing number of vertices, $|\mathbf{L}| = 2, \langle k \rangle \approx \sqrt{|V_M|}$ (bottom left), **c** increasing number of layers, $|V| = 1000, \langle k \rangle \approx 4$ (top right), and **d** increasing number of layers, $|V| = 1000, \langle k \rangle \approx \sqrt{|V_M|}$ (bottom right). The error bars in black indicate standard deviation over four runs

Table 6 Layer aggregation operator performance

Dataset	multinet	MuxViz	Pymnet	Py3plex
cs-aarhus	0.0 (0.0)	0.2 (0.2)	0.0 (0.0)	0.0 (0.0)
London-transport	0.0 (0.0)	0.1 (0.0)	0.0 (0.0)	0.0 (0.0)
Euair-transport	0.0 (0.0)	0.1 (0.0)	0.0 (0.0)	0.2 (0.0)
Friendfeed-twitter	–	31.5 (1.6)	309.8 (15.4)	96.0 (1.2)
Friendfeed	–	47.6 (0.8)	559.2 (14.4)	168.4 (1.1)

Rounded values in seconds. We report the average performance and standard deviation (in parentheses) over four runs. A dash indicates that the task did not complete within the timeout

In Fig. 3 we note performance differences between the software when interactively updating the network, sometimes in orders of magnitude. This difference is particularly apparent in software for different programming languages: we note the sharp differences in performance between Python (Pymnet, Py3plex) and R (Multinet, MuxViz) software. However, we note that all software follows a similar exponential curve when adding and removing edges to the network. This suggests that the observed performance differences can be attributed to software design choices other than the data structure.

Layer aggregation

Similarly to the network loading operator, in Fig. 4 we can observe that the number of edges increasing heavily affects the performance of the different implementations for the layer aggregation task. This trend is also particularly visible in Table 6, as Multinet is not able to load and aggregate the large social networks within the timeout, despite being able to process synthetic networks with a greater amount of nodes. We can also observe that adjacency matrix-based MLN representations (MuxViz) appear to perform better, sometimes faster by an order of magnitude, except when aggregating a large number of layers. Finally, for networks of smaller sizes, we note that the difference in execution times between the tested libraries is minimal.

Discussion

Based on the previous two sections, we can now summarize the status of MLN engineering as seen through current MLN software. We do this by first providing an overview of the area, organised along four main points, in Sect. MLN engineering: an overview. Building on this overview and on the limitations we identify in the area, in Sect. Future explorations we discuss possible future directions.

MLN engineering: an overview

Interoperability As previously discussed, not all features of the MLN model are universally supported within the software covered. The algorithmic support for features such as interlayer edges and multiple aspects varies, and the same applies to vertex, node and edge attributes. This effectively limits the development and implementation of metadata-enriched methods, e.g. for layer manipulation or community detection. In conjunction with a low coverage for layer manipulation operators, all of the previous restrict our ability to visually explore alternative MLNs interactively (McGee et al. 2021).

We also note the lack of a common file input/output format for MLNs. While there is general support for the multilayer edgelist as input, the format offers limited support for node metadata.

There is also a limited overlap of operators between the selected software. Looking at our taxonomy in Sect. Models and operators, we observe that most of the software implement a few common operators (e.g. reading, generating, aggregating and visualizing networks) along with structural measures like node degree and centrality. However, each library typically specializes in different types of operators. For example, MuxViz contains operators for dynamics and various network metrics, Pymnet implements operators for isomorphisms and multiple MLN variants for clustering coefficients, while Multinet provides more options for layer comparison and clustering.

Overall, there is little interoperability between MLN software programs, which can cause practical problems for researchers using MLNs, as they need to become familiar with the specific software representation to reach their expected research objective (Kinsley et al. 2020; Finn 2021). This can be compared both with relational database systems, where anyone familiar with the relational model can easily switch from one system to the other when it comes to using their core functionality, and also with graph analysis systems, where switching, for example, from NetworkX to igraph, only involves some

syntactic differences when we focus on core graph operators (generators, basic centrality measures, etc.).

Data definition and manipulation Our taxonomy is also useful when looking at what operators are missing. There is no consensus on what should be considered a basic (or necessary) operator to manipulate MLNs, and there is a lack of such operators. As a consequence, it is not possible to express complex portable queries, e.g. generating different views and aggregations from the same initial dataset and sharing the definition of the manipulation process instead of the resulting data.

At the same time, there is no support for constraints to control the validity of the data, nor indices, to control the balance between time and space efficiency. This also makes it difficult to formulate complex queries.

Large networks As we can observe from the scalability tests in the experimental study, there are general issues handling very large networks. Not surprisingly, network size and density affect the performance of the tested operators. With the number of vertices in the network increasing, even with low average degrees no software is able to load the network within the timeout. Often, the processes are killed due to not enough memory being available. Considering modern data sources can be very large in size (e.g. population networks (Kazmina et al. 2023; Bokányi et al. 2023), containing tens of millions of nodes), this further points towards the need for a discussion about the performance of different data structures for MLNs.

In addition, the experiments suggest that none of the featured implementations can consistently outperform the rest, as all have their own advantages and disadvantages. An adjacency matrix representation, for example, performs better for operators like layer aggregation, and edges can be quickly added and removed, while adding and removing nodes or layers might require recreating the object, as the matrix structure becomes different. Adjacency list-based data structures are able to handle large networks, but they appear to be less efficient for layer aggregation. On the other hand, using native graph objects from other libraries to represent layers can often be efficient, but it does not necessarily scale well as the number of layers increases, nor does it efficiently support layer transformation capabilities. The choice of data structure affects which operators can be more or less easily implemented in the first place, and their efficiency.

Benchmarks A major consequence of the previous issues is a difficulty designing systematic experimental comparisons, including the one presented in this work. First, the lack of a common standard makes it hard to decide what should be deemed a necessary operator when designing an MLN system. Second, it is equally difficult to establish a fair baseline for comparison of the various operators, also considering the variety in data structure choices. As a result, this slows down the process of identifying operators in need of optimization.

If we look at research areas focused on the development of data management and analysis pipelines, the above considerations have been critical. Looking at the relational database meta-model, we have a clear understanding of what operators can be applied to the data, and also practical knowledge of how different operators behave based on their implementations (e.g. data structures) and data distributions, through well-established benchmarks.

Future explorations

The overview of MLN engineering presented in the previous section suggests that the field is ready for the development of effective data manipulation and analysis systems. Several MLN systems exist, and when considered together they do provide a lot of the required functionality. However, looking at typical criteria used when selecting software components, like interoperability (Carvallo Vega et al. 2007), the field clearly requires additional efforts. Similarly, there are little considerations for other important characteristics, such as the security and functionality compliance of the software. These could be critical, for example, when analyzing MLNs from registry or personal data. Considering the usefulness and increasing popularity of MLNs in analyzing complex data, future steps in the field should aim to address these challenges.

Our overview highlights a critical area of exploration: addressing data management needs for MLN software. Considering the challenges when processing large MLNs, a potential solution includes designing a database management system-based model to store and manipulate MLN data. Such a model should then be integrated into MLN analysis software for testing. Implementing data management models inspired by well-established ones, such as those used in relational and graph databases, into MLN systems, can also help alleviate potential security and functionality compliance concerns.

Also relevant is the definition of a set of essential operators (i.e. a query language), complemented with a clear definition of operator behaviour. Such a set of operators would allow for improved interoperability between software, easier formulation of complex queries and definition of constraints for MLNs. Given a set of basic operators, we can then consider a common basis for experimental comparisons of software, data structures and novel operator extensions. A set of operators alone does not make the task of comparing software trivial, as it is not easy to define a single comparative indicator without considering other design choices in the software. However, this process will simplify the identification of operators and queries that can be optimized. In summary, such a language would address several of the limitations we highlighted in the previous section.

MLN engineering as a field will still benefit from new theoretical developments. Examples of areas where we foresee new developments include computationally expensive analytical tasks like community detection and classification, where modern approaches used in machine learning (such as graph neural networks and embeddings) have started appearing in the MLN literature, but are still not as developed and especially accessible as in other areas. A related challenge is also defining methods for MLNs considering attributes, for example aggregating or slicing layers based on the values of a node attribute. At the same time, this study suggests how theoretical developments have so far lacked corresponding efforts to engineer them into usable systems.

Conclusion

With the popularity of MLNs for storage, manipulation and analysis of complex systems, it is imperative to question whether the MLN engineering landscape is mature enough to handle challenges posed by modern data sources. In this paper, we delve into these questions by looking at currently available MLN analysis software. We provide a taxonomy of MLN manipulation and analysis operators featured in the software included in this study, and experimentally study common operators. Based on these analyses, we

then discuss the current status and limitations of MLN engineering as a distinct research area.

We find that the current MLN engineering ecosystem consists of multiple software implementations with limited interoperability between them. For example, there is a relatively small number of common operators within the featured software, along with a variety of underlying implementations, and a limited support for vertex, node and edge metadata. This lack of a common baseline creates practical problems not only for researchers using MLNs, but also when designing benchmark studies. We also note the issues software currently faces when processing large MLNs.

Considering the popularity of MLNs, future work in the MLN engineering field should aim to address these limitations, in order to improve usability of MLN systems. A major future direction includes the design of systems capable of efficiently processing large MLN data. This can potentially be achieved via integrating relational- or graph-database management systems into MLN analysis software, in order to be able to process large networks. As our experimental study focuses primarily on the manipulation operators, it can potentially be extended to compare the scalability of relational and graph database management-based models for MLNs. Other major future directions include the definition of a query language for MLNs, the design of larger-scale benchmarks including more datasets and operators, also on settings with more computational resources. In turn, this can help highlighting potential areas for optimization. Finally, another promising direction is the theoretical definition and testing of novel methods for computationally expensive MLN tasks, such as community detection or layer manipulation.

Acknowledgements
The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at Chalmers Centre for Computational Science and Engineering (C3SE), High Performance Computing Center North (HPC2N) and Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) partially funded by the Swedish Research Council through grant agreement no. 2022-06725. We would like to thank the authors of the featured software, Manlio De Domenico, Mikko Kivelä, Blaž Škrlj, Pietro Monticone and Claudio Moroni, for their correspondence and insightful feedback on early versions of the manuscript. Finally, we are grateful to the anonymous reviewers for their helpful remarks.

Author contributions
G.P., M.M. and B.P. designed the study. G.P. performed the feature comparison and experiments, and led writing of the paper. All authors revised and approved the final manuscript.

Funding
Open access funding provided by Uppsala University. This work has been partly funded by eSSENCE, an e-Science collaboration funded as a strategic research area of Sweden. M.M. has been partly funded by EU CEF grant number 2394203 (NORDIS - NORdic observatory for digital media and information DISorder). We also acknowledge support from the French Institute in Sweden (IFS).

Availability of data and materials
The scripts for the experimental comparison, including the synthetic network generator, are available on https://github.com/uuinfolab/paper.24_ApplNetSci_MLN-Engineering-Challenges. All other datasets used are publicly available; see Sect. Datasets for references.

Declarations

Competing interests
The authors declare no competing interest.

Received: 2 April 2024 Accepted: 25 November 2024
Published online: 10 December 2024

References

Aleta A, Meloni S, Moreno Y (2017) A Multilayer perspective for the analysis of urban transportation systems. Scientific Reports 7(1):44359. Number: 1 Publisher: Nature Publishing Group. Accessed 2023-10-20 https://doi.org/10.1038/srep44359

Angles R, Hogan A, Lassila O, Rojas C, Schwabe D, Szekely P, Vrgoč D (2022) Multilayer graphs: a unified data model for graph databases. In: Proceedings of the 5th ACM SIGMOD joint international workshop on graph data management experiences & systems (GRADES) and network data analytics (NDA). GRADES-NDA '22, pp. 1–6. Association for computing machinery, New York, NY, USA. https://doi.org/10.1145/3534540.3534696. Accessed 2022-09-27

Auber D, Archambault D, Bourqui R, Delest M, Dubois J, Lambert A, Mary P, Mathiaut M, Melançon G, Pinaud B, Renoust B, Vallet J (2017) Tulip 5. In: Alhajj R, Rokne J (eds) Encyclopedia of social network analysis and mining. Springer, New York, NY, pp 1–28. https://doi.org/10.1007/978-1-4614-7163-9_315-1

Bastian M, Heymann S, Jacomy M (2009) Gephi: an open source software for exploring and manipulating networks. In: Proceedings of the international AAAI conference on web and social media 3(1):361–362. Accessed 2023-08-30 https://doi.org/10.1609/icwsm.v3i1.13937

Berlingerio M, Coscia M, Giannotti F, Monreale A, Pedreschi D (2011) Foundations of multidimensional network analysis. In: 2011 International conference on advances in social networks analysis and mining, pp. 485–489. IEEE, Kaohsiung City, Taiwan. https://doi.org/10.1109/ASONAM.2011.103 . http://ieeexplore.ieee.org/document/5992618/ Accessed 2023-08-30

Bianconi G (2022) Multilayer networks: structure and function. Oxford University Press, Oxford, New York

Boccaletti S, Bianconi G, Criado R, Genio CI, Gómez-Gardeñes J, Romance M, Sendiña-Nadal I, Wang Z, Zanin M (2014) The structure and dynamics of multilayer networks. Phys Rep 544(1):1–122. https://doi.org/10.1016/j.physrep.2014.07.001. (Accessed 2022-11-30)

Bokányi E, Heemskerk EM, Takes FW (2023) The anatomy of a population-scale social network. Scientific Reports 13(1):9209. Nature Publishing Group. Accessed 2023-08-24 https://doi.org/10.1038/s41598-023-36324-9

Bokanyi E, Jong R, Zoete B, Kazmina Y (2022) POPNET multi layered network library. https://github.com/popnet-io/popnet_mln

Borgatti SP, Mehra A, Brass DJ, Labianca G (2009) Network analysis in the social sciences. Science 323(5916):892–895. https://doi.org/10.1126/science.1165821. (Accessed 2024-03-23)

Bott H (1928) Observation of play activities in a nursery school. Genet Psychol Monogr 4:44–88

Brandes U, Eiglsperger M, Herman I, Himsolt M, Marshall MS (2002) GraphML progress report structural layer proposal. In: Mutzel P, Jünger M, Leipert S (eds) Graph Drawing. Springer, Berlin, pp 501–512. https://doi.org/10.1007/3-540-45848-4_59

Brehmer M, Munzner T (2013) A multi-level typology of abstract visualization tasks. IEEE Trans Visual Comput Graphics 19(12):2376–2385. https://doi.org/10.1109/TVCG.2013.124. (Accessed 2024-03-23)

Brodka P, Chmiel A, Magnani M, Ragozini G (2018) Quantifying layer similarity in multiplex networks: a systematic study. R Soc Open Sci 5(8):171747

Cardillo A, Gómez-Gardeñes J, Zanin M, Romance M, Papo D, Pozo Fd, Boccaletti S (2013) Emergence of network features from multiplexity. Scientific Reports 3, 1344 https://doi.org/10.1038/srep01344 . Accessed 2022-12-07

Carvallo Vega JP, Franch Gutiérrez J, Quer C (2007) Determining criteria for selecting software components: lessons learned. IEEE Softw 24(3):84–94. https://doi.org/10.1109/MS.2007.70

Celli F, Di Lascio FML, Magnani M, Pacelli B, Rossi L (2010) Social network data and practices: the case of friendfeed. In: Chai S-K, Salerno JJ, Mabry PL (eds) Advances in social computing lecture notes in computer science. Springer, Berlin, pp 346–353. https://doi.org/10.1007/978-3-642-12079-4_43

Chen C, Yan X, Zhu F, Han J, Yu PS (2009) Graph OLAP: a multi-dimensional framework for graph data analysis. Knowl Inf Syst 21(1):41–63. https://doi.org/10.1007/s10115-009-0228-9. (Accessed 2023-11-29)

Coscia M (2022) Generalized Euclidean Measure to Estimate Distances on Multilayer Networks. ACM Trans Knowl Discov Data 16(6):1–22. https://doi.org/10.1145/3529396. (Accessed 2023-08-30)

Cuenca E, Sallaberry A, Ienco D, Poncelet P (2022) VERTIGo: a visual platform for querying and exploring large multilayer networks. IEEE Trans Visual Comput Graphics 28(3):1634–1647. https://doi.org/10.1109/TVCG.2021.3067820

De Domenico M (2017) Multilayer modeling and analysis of human brain networks. GigaScience 6(5):1–8. https://doi.org/10.1093/gigascience/gix004. (Accessed 2022-12-07)

De Domenico M, Solé-Ribalta A, Gómez S, Arenas A (2014) Navigability of interconnected networks under random failures. Proc Natl Acad Sci USA 111(23):8351–8356. https://doi.org/10.1073/pnas.1318469111. (Accessed 2022-12-07)

De Domenico M, Porter MA, Arenas A (2015) MuxViz: a tool for multilayer analysis and visualization of networks. J Complex Netw 3(2):159–176. https://doi.org/10.1093/comnet/cnu038. (Accessed 2022-12-07)

Dickison ME, Magnani M, Rossi L (2016) Multilayer social networks. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9781139941907 . https://www.cambridge.org/core/books/multilayer-social-networks/39383306D9843313057CECEBF7B9BF26 Accessed 2023-10-23

Espinosa-Rada A (2023) netmem: social network measures using matrices. original-date: 2020-04-26T14:04:09Z. https://github.com/anespinosa/netmem Accessed 2023-08-30

Feyer SP, Pinaud B, Kobourov S, Brich N, Krone M, Kerren A, Behrisch M, Schreiber F, Klein K (2023) 2D, 2.5 D, or 3D? an exploratory study on multilayer network visualisations in virtual reality. IEEE Transactions on Visualization and Computer Graphics. IEEE. Accessed 2024-03-23

Finn KR (2021) Multilayer network analyses as a toolkit for measuring social structure. Current Zool 67(1):81–99. https://doi.org/10.1093/cz/zoaa079. (Accessed 2023-08-30)

Finn KR, Silk MJ, Porter MA, Pinter-Wollman N (2019) The use of multilayer network analysis in animal behaviour. Anim Behav 149:7–22. https://doi.org/10.1016/j.anbehav.2018.12.016. (Accessed 2023-08-24)

Frydman N, Freilikhman S, Talpaz I, Pilosof S (2023) Practical guidelines and the EMLN R package for handling ecological multilayer networks. EcoEvoRxiv. Accessed 2023-08-30

Galimberti E, Bonchi F, Gullo F, Lanciano T (2020) Core decomposition in multilayer networks: theory, algorithms, and applications. ACM Trans Knowl Discov Data 14(1):11–11140. https://doi.org/10.1145/3369872. (Accessed 2023-08-30)
Gallotti R, Barthelemy M (2014) Anatomy and efficiency of urban multimodal mobility. Sci Rep 4:6911. https://doi.org/10.1038/srep06911. (Accessed 2022-12-07)
Ghawi R, Pfeffer J (2022) A community matching based approach to measuring layer similarity in multilayer networks. Social Netw 68:1–14. https://doi.org/10.1016/j.socnet.2021.04.004. (Accessed 2024-06-03)
Gibson RA, Mucha PJ (2022) Finite-state parameter space maps for pruning partitions in modularity-based community detection. Sci Rep 12(1):15928. https://doi.org/10.1038/s41598-022-20142-6
Giordano G, Ragozini G, Vitale MP (2019) Analyzing multiplex networks using factorial methods. Social Netw 59:154–170. https://doi.org/10.1016/j.socnet.2019.07.005. (Accessed 2024-06-03)
Hammoud Z, Kramer F (2018) mully: an R package to create modify and visualize multilayered graphs. Genes 9(11):519. https://doi.org/10.3390/genes9110519
Hammoud Z, Kramer F (2020) Multilayer networks: aspects, implementations, and application in biomedicine. Big Data Analytics 5(1):2. https://doi.org/10.1186/s41044-020-00046-0. (Accessed 2023-08-24)
Hanteer O, Rossi L, D'Aurelio DV, Magnani M (2018) From interaction to participation: the role of the imagined audience in social media community detection and an application to political communication on twitter. In: 2018 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM), pp. 531–534. https://doi.org/10.1109/ASONAM.2018.8508575 . ISSN: 2473-991X. https://ieeexplore.ieee.org/document/8508575 Accessed 2024-06-05
Heer J, Perer A (2014) Orion: a system for modeling, transformation and visualization of multidimensional heterogeneous networks. Inf Vis 13(2):111–133. https://doi.org/10.1177/1473871612462152. (Accessed 2022-11-14)
Hoe-Lian GD, Chua A, Anqi KD, Boon-Hui KE, Bok-Tong ME, Wen-Min NM (2006) A checklist for evaluating open source digital library software. Online information review 30(4):360–379. Emerald Group Publishing Limited. Accessed 2024-06-05. Publisher: Emerald Group Publishing Limited. Accessed 2024-06-05 https://doi.org/10.1108/14684520610686283
Hristova D, Noulas A, Brown C, Musolesi M, Mascolo C (2016) A multilayer approach to multiplexity and link prediction in online geo-social networks. EPJ Data Science 5(1):1–17. SpringerOpen. Accessed 2024-06-05 https://doi.org/10.1140/epjds/s13688-016-0087-z
Interdonato R, Tagarelli A, Ienco D, Sallaberry A, Poncelet P (2017) Local community detection in multilayer networks. Data Min Knowl Disc 31(5):1444–1479. https://doi.org/10.1007/s10618-017-0525-y. (Accessed 2023-08-30)
Interdonato R, Magnani M, Perna D, Tagarelli A, Vega D (2020) Multilayer network simplification: approaches, models and methods. Comput Sci Rev 36:100246. https://doi.org/10.1016/j.cosrev.2020.100246. (Accessed 2023-01-24)
Jeub LGS, Bazzi M, Jutla IS, Mucha PJ (2019) A generalized Louvain method for community detection implemented in MATLAB. GenLouvain. original-date: 2016-11-25T14:49:08Z. https://github.com/GenLouvain/GenLouvain Accessed 2023-08-30
Kazmina Y, Heemskerk EM, Bokanyi E, Takes FW (2023) Socio-economic Segregation in a Population-scale social network. arXiv:2305.02062 [physics]. Accessed 2023-08-25
Kinsley AC, Rossi G, Silk MJ, VanderWaal K (2020) Multilayer and multiplex networks: an introduction to their use in veterinary epidemiology. Frontiers in Veterinary Science 7. Accessed 2023-08-24
Kivelä, M.: Multilayer Networks Library for Python (Pymnet) - Multilayer Networks Library 0.1 documentation. http://www.mkivela.com/pymnet/ Accessed 2023-02-19
Kivela M, Arenas A, Barthelemy M, Gleeson JP, Moreno Y, Porter MA (2014) Multilayer networks. J Complex Netw 2(3):203–271. https://doi.org/10.1093/comnet/cnu016. (Accessed 2022-11-30)
Kouvaris NE, Hata S, Guilera AD- (2015) Pattern formation in multiplex networks. Sci Rep 5(1):10840. https://doi.org/10.1038/srep10840
Kriegel H-P, Schubert E, Zimek A (2017) The (black) art of runtime evaluation: are we comparing algorithms or implementations? Knowl Inf Syst 52(2):341–378. https://doi.org/10.1007/s10115-016-1004-2. (Accessed 2023-12-13)
Magnani M, Rossi L (2013) Formation of multiple networks. In: Greenberg AM, Kennedy WG, Bos ND (eds) Social computing, behavioral-cultural modeling and prediction. Springer, Berlin, pp 257–264. https://doi.org/10.1007/978-3-642-37210-0_28
Magnani M, Hanteer O, Interdonato R, Rossi L, Tagarelli A (2021) Community detection in multiplex networks. ACM Comput Surv. https://doi.org/10.1145/3444688
Magnani M, Rossi L, Vega D (2021) Analysis of Multiplex Social Networks with R. J Stat Softw 98:1–30. https://doi.org/10.18637/jss.v098.i08. (Accessed 2022-12-05)
Magnani M, Micenkova B, Rossi L (2013) Combinatorial analysis of multiple networks. arXiv. arXiv:1303.4986 [physics]. Accessed 2023-08-24
Magnani M, Rossi L (2011) The ML-model for multi-layer social networks. In: Proceedings - 2011 International conference on advances in social networks analysis and mining, ASONAM 2011. https://doi.org/10.1109/ASONAM.2011.114
Matter D, Kuznetsova E, Vziatysheva V, Vitulano I, Pfeffer J (2023) Temporally stable multilayer network embeddings: a longitudinal study of Russian propaganda. arXiv. arXiv:2307.10264 [cs]. Accessed 2024-03-26
McGee F, Ghoniem M, Melançon G, Otjacques B, Pinaud B (2019) The state of the art in multilayer network visualization. Computer graphics forum 38(6), 125–149 https://doi.org/10.1111/cgf.13610. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13610. Accessed 2022-11-30
McGee F, Ghoniem M, Otjacques B, Renoust B, Archambault D, Kerren A, Pinaud B, Melançon G, Pohl M, Landesberger T (2021) Visual analysis of multilayer networks. Synthesis lectures on visualization. Springer, Cham. https://doi.org/10.1007/978-3-031-02608-9 . Accessed 2022-11-30
Mondal S, Basu A, Mukherjee N (2020) Building a trust-based doctor recommendation system on top of multilayer graph database. J Biomed Inform 110:103549. https://doi.org/10.1016/j.jbi.2020.103549. (Accessed 2023-06-11)
Moreno JL, Jennings HH (1934) Who shall survive?: A new approach to the problem of human interrelations. Nervous and Mental Disease Publishing Co., Washington, D. C

Moroni C, Monticone P (2023) MultilayerGraphs.jl: Multilayer network science in Julia. J Open Source Softw 8(83):5116. https://doi.org/10.21105/joss.05116
Nurmi T, Badie-Modiri A, Coupette C, Kivelä M (2024) pymnet: A python library for multilayer networks. J Open Source Softw 9(99):6930. https://doi.org/10.21105/joss.06930
Ostoic JAR (2020) Algebraic analysis of multiple social networks with multiplex. J Stat Softw 92:1–41. https://doi.org/10.18637/jss.v092.i11. (Accessed 2023-08-30)
Peixoto TP (2014). The graph-tool python library figshare https://doi.org/10.6084/m9.figshare.1164194.v14. https://figshare.com/articles/dataset/graph_tool/1164194/14 Accessed 2023-08-30
Perna D, Interdonato R, Tagarelli A (2018) Identifying users with alternate behaviors of lurking and active participation in multilayer social networks. IEEE Trans Comput Social Syst 5(1):46–63. https://doi.org/10.1109/TCSS.2017.2762730
Pilosof S, Porter MA, Pascual M, Kéfi S (2017) The multilayer nature of ecological networks. Nat Ecol Evol 1(4):0101. https://doi.org/10.1038/s41559-017-0101
Renoust B, Melançon G, Munzner T (2015) Detangler: visual analytics for multiplex networks. Comput Graphics Forum 34(3):321–330. https://doi.org/10.1111/cgf.12644. (Accessed 2022-11-14)
Robitaille AL, Webber QMR, Turner JW, Vander Wal E (2021) The problem and promise of scale in multilayer animal social networks. Current Zool 67(1):113–123. https://doi.org/10.1093/cz/zoaa052. (Accessed 2023-08-30)
Rossi L, Magnani M (2015) Towards effective visual analytics on multiplex and multilayer networks. Chaos, Solitons and Fractals. https://doi.org/10.1016/j.chaos.2014.12.022
Santana J, Hoover R, Vengadasubbu M (2017) Investor commitment to serial entrepreneurs: a multilayer network analysis. Soc Netw 48:256–269. https://doi.org/10.1016/j.socnet.2016.10.002. (Accessed 2024-03-26)
Santra A, Komar K, Bhowmick S, Chakravarthy S (2022) From base data to knowledge discovery - a life cycle approach - using multilayer networks. Data & Knowl Eng 141:102058. https://doi.org/10.1016/j.datak.2022.102058. (Accessed 2023-02-20)
Shi C, Li Y, Zhang J, Sun Y, Yu PS (2017) A survey of heterogeneous information network analysis. IEEE Trans Knowl Data Eng 29(1):17–37. https://doi.org/10.1109/TKDE.2016.2598561
Škrlj B, Kralj J, Lavrač N (2019) Py3plex: a library for scalable multilayer network analysis and visualization. In: Aiello LM, Cherifi C, Cherifi H, Lambiotte R, Lió P, Rocha LM (eds) Complex networks and their applications VII studies in computational intelligence. Springer, Cham, pp 757–768. https://doi.org/10.1007/978-3-030-05411-3_60
Steer B, Arnold N, Ba CT, Lambiotte R, Yousaf H, Jeub L, Murariu F, Kapoor S, Rico P, Chan R, Chan L, Alford J, Cuadrado RGCF, Barnes MR, Zhong P, Biyong JNP, Alnaimi A (2023) Raphtory: The temporal graph engine for Rust and Python. arXiv. arXiv:2306.16309. Accessed 2023-08-25
Sun Y, Han J, Yan X, Yu PS, Wu T (2022) Heterogeneous information networks: the past, the present, and the future. In: Proceedings of the VLDB endowment vol. 15, no. 12, pp. 3807–3811. https://doi.org/10.14778/3554821.3554901. (Accessed 2023-11-29)
Szárnyas G, Kővári Z, Salánki A, Varró D (2016) Towards the characterization of realistic models: evaluation of multidisciplinary graph metrics. In: Proceedings of the ACM/IEEE 19th international conference on model driven engineering languages and systems. MODELS '16, pp. 87–94. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2976767.2976786 . Accessed 2023-08-30
Timme N, Ito S, Myroshnychenko M, Yeh F-C, Hiolski E, Hottowy P, Beggs JM (2014) Multiplex networks of cortical and hippocampal neurons revealed at different timescales. PLoS ONE 9(12):115764. https://doi.org/10.1371/journal.pone.0115764. (Accessed 2022-12-07)
Timóteo S, Correia M, Rodríguez-Echeverría S, Freitas H, Heleno R (2018) Multilayer networks reveal the spatial structure of seed-dispersal interactions across the Great Rift landscapes. Nat Commun 9:140. https://doi.org/10.1038/s41467-017-02658-y. (Accessed 2022-12-07)
Traxl D, Boers N, Kurths J (2016) Deep graphs-a general framework to represent and analyze heterogeneous complex systems across scales. Chaos: An Interdiscip J Nonlinear Sci 26(6):065303. https://doi.org/10.1063/1.4952963. (Accessed 2023-08-30)
Trimbour R, Deutschmann IM, Cantini L (2023) Molecular mechanisms reconstruction from single-cell multi-omics data with HuMMuS. bioRxiv. Pages: 2023.06.09.543828 Section: New Results. https://doi.org/10.1101/2023.06.09.543828 . https://www.biorxiv.org/content/10.1101/2023.06.09.543828v1 Accessed 2023-08-30
Ustek-Spilda F, Vega D, Magnani M, Rossi L, Shklovski I, Lehuede S, Powell A (2021) A twitter-based study of the European Internet of Things. Inf Syst Front 23(1):135–149. https://doi.org/10.1007/s10796-020-10008-5. (Accessed 2023-03-01)
Vaiana M, Muldoon SF (2020) Multilayer brain networks. J Nonlinear Sci 30(5):2147–2169. https://doi.org/10.1007/s00332-017-9436-8. (Accessed 2024-06-05)
Vijayaraghavan VS, Noël P-A, Maoz Z, D'Souza RM (2015) Quantifying dynamical spillover in co-evolving multiplex networks. Sci Rep 5(1):15142. https://doi.org/10.1038/srep15142
Wehmuth K, Fleury E, Ziviani A (2016) On multiaspect graphs. Theoretical Comput Sci 651:50–61. https://doi.org/10.1016/j.tcs.2016.08.017
Xia J, Gill EE, Hancock REW (2015) Network analyst for statistical, visual and network-based meta-analysis of gene expression data. Nature Protocols 10(6):823–844. https://doi.org/10.1038/nprot.2015.052
Zitnik M, Leskovec J (2017) Predicting multicellular function through multi-layer tissue networks. Bioinformatics 33(14):190–198. https://doi.org/10.1093/bioinformatics/btx252. (Accessed 2023-08-30)

Publisher's Note