MLN-geeWhiz: Supporting Complex Data Analysis Including Visualization

Amey Shinde[1], Viraj Sabhaya[1], Kevin Farokhrouz[1], Fariba Irany[2], Ali Khan[2], Sanjukta Bhowmick[2], Abhishek Santra[1(✉)], and Sharma Chakravarthy[1]

[1] IT Laboratory and CSE Department, UT Arlington, Arlington, USA
{amey.shinde,virajvipinbhai.sabhaya,kaf2886, abhishek.santra}@mavs.uta.edu, sharmac@cse.uta.edu
[2] University of North Texas, Denton, USA
{FaribaAfrinIrany,AliKhan}@my.unt.edu, sanjukta.bhowmick@unt.edu

Abstract. The availability of numerous algorithms and techniques developed for complex data analysis makes it difficult for domain users to navigate the life cycle workflow for their analysis. One way to overcome this is by providing interactive tools that are intuitive to use, graphical, and can be used for all stages of the life cycle by non-experts – from modeling to complex analysis to drill down & visualization. If these tools are also extensible and modular, they allow additions of new models and algorithms as they become available for the benefit of the larger community.

Graphs have been extensively used to study the complex systems of interacting entities from diverse disciplines. However, when studying complex data with multiple types of entities and relationships, simple or even attributed graphs are not always ideal for modeling them. Currently, to derive knowledge from complex data with multiple entity types and relationships, multilayer networks (or MLNs) are being increasingly used.

MLN-geeWhiz, is conceived to be such an end-to-end interactive tool with its initial version available for public use. Its purpose is to fill a void and allow non-experts from diverse disciplines to analyze their complex data using state-of-the-art analysis techniques and visualize the results. This dashboard has been developed to use complex analysis techniques with ease. It is modular, so new algorithms and approaches can be added transparently. The user can upload, generate needed MLNs (or graphs), analyze them and visualize results in many ways. In this paper, we discuss the challenges of a modular architecture needed for supporting complex analysis underneath and visualization to understand the results.

Keywords: Multilayer Networks · Modular/Extensible Architecture · Complex Data Analysis · Visualization

1 Motivation

Unlike modeling a relational or object-oriented database where a schema is generated first and then populated for analysis, in the graph models, the graph itself contains data. The schema is implicit. Hence, all the necessary (and hopefully minimal) information

A. Dasgupta et al. (Eds.): BDA 2024, LNCS 15526, pp. 219–239, 2025.
https://doi.org/10.1007/978-3-031-81821-9_13

for analysis need to be included in the graphs generated. Typically, nodes of the graph represent entities and edges represent relationships. Although labels can be used both for nodes and edges, it is important to keep the amount of information minimal to satisfy the objectives specified. For example, if you are clustering authors, an author can be represented by an ID and not include other attributes which are not needed for analysis. Similarly, for edges. However, the rest of the information of entities and relationships is needed for drill-down in order to visualize results for understand. We focus on the MLN graph model, its analysis, and visualization in this paper.

1.1 Multilayer Networks and Their Utility

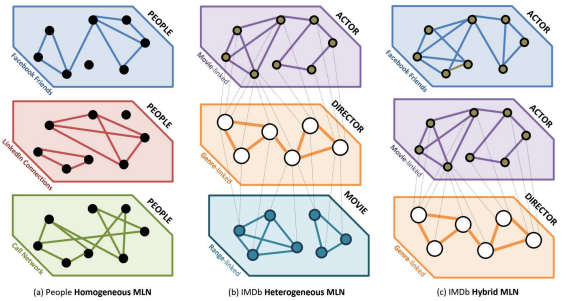


Fig. 1. MLN Types

Multilayer networks (MLNs) provide an alternate way to model complex data sets. It can capture each relationship as a layer which is a separate simple graph increasing understandability. It also affords itself for flexible analysis of layers. Nodes from different layers can also be connected by an edge if relationships exist between two entity types. For example, actor and director layers can be connected by the 'direct-actor' relationship where a director entity has an edge with each actor entity s/he has directed in a movie. Design of MLN model for a data set has been addressed separately in the literature [37]. MLNs, by their structure, also offer flexibility to analyze each layer individually (and in parallel) as well as arbitrary subsets of layer combinations. Based on entity types in each layer and connectivity within and across layers, MLNs can be classified as homogeneous MLNs (HoMLNs), heterogeneous MLNs (HeMLNs), and hybrid MLNs (HyMLNs), illustrated in Fig. 1(a), (b), and (c), respectively.

Definition 1. *A multilayer network $MLN(G, X)$, is defined by two sets of graphs. The set $G = \{G_1, G_2, \ldots, G_n\}$ contains **simple** graphs (one for each layer), where $G_i(V_i, E_i)$ is defined by a set of vertices V_i and a set of edges E_i. An edge $e(v, u) \in E_i$, connects vertices v and u, where $v, u \in V_i$. The set $X = \{X_{1,2}, X_{1,3}, \ldots, X_{m-1,m}\}$ consists of **bipartite** graphs[1]. Each graph $X_{i,j}(V_i, V_j, L_{i,j})$ is defined by two sets of vertices V_i and V_j and a set of edges (or links) $L_{i,j}$, such that for every link $l(a, b) \in L_{i,j}$, $a \in V_i$ and $b \in V_j$, where V_i (V_j) is the vertex set of graph G_i (G_j). Some of $X_{i,j}$ can be empty.*

MLNs find use in diverse disciplines that require modeling complex interactions, including drug design [21], understanding collaboration patterns [17], understanding

[1] The number of bipartite graphs can be less or even more than the number of graphs/layers. That is, m can be $<=$ or even $> n$.

the interaction between spoken and written language [24], discovering long-term coexistence of viruses [33], comparing the evolution of species [43], finding vulnerabilities in power grids [29], and identifying illegal activities via social interactions [14]. More recently, MLNs have been used to model the spread of COVID-19, where each layer represents a different social interaction [39] and how international supply chains have been affected by the restrictions imposed due to COVID-19 [20].

The use of MLNs for modeling is a new frontier for data analysis. Analysis algorithms and even the definitions of metrics, such as communities, centrality, substructures, and k-core are still evolving. Thus, a platform where researchers can access the latest algorithms and apply them to their data sets is essential for advancing this field. Although the analysis of multilayer networks is a fast-growing area, there is yet no tool for the generation, analysis, and visualization of MLNs, let alone the sharing of data and results or community interaction.

Problem Addressed: Given the complexity of complex data analysis workflow as shown in Fig. 2 and the availability of numerous algorithms (as shown in Table 1) for analysis of MLNs (as elaborated in Sect. 2), develop a *modular and extensible* interactive tool that will support the life cycle. In addition to modeling and analysis, the tool should also support *drill-down of results & visualization*.

The contributions of this paper are:

- An **interactive, web-based dashboard**[2] supporting life cycle of complex data analysis using graphs and MLN data models.
- Generation of **graphs and MLNs**, their analysis, drill-down, and visualization.
- **Extensibility** of modules to extend layer generation, analysis, and visualization.
- **Modular** approach for concurrent development by groups with different skill sets.
- **Multiple (interactive) Visualization Alternatives** for base and analyzed results.
- Support for **multiple concurrent users** and **Optimization** of response time

This paper is organized as follows. Section 2 discusses analysis life cycle, MLN analysis alternatives, and a *sample* of algorithms to demonstrate the vastness of the analysis space. Section 3 discusses related work. Section 4 discusses the challenges to be addressed during the design of the interactive tool (**MLN-geeWhiz** dashboard.) Sect. 5 details the architecture of the dashboard and the front-end design. Section 6 presents critical back-end modules of this dashboard: layer generator, graph and MLN analysis, and drill-down & visualization. Dashboard's usage on some real-world data sets are highlighted in Sect. 7. Conclusions are in Sect. 8.

2 Life Cycle of Complex Data Analysis

In contrast to data mining, big data analysis is more holistic and far more complex. Multiple types of analysis need to be performed based on the data domain and analysis objectives which may vary significantly from one data set to another. Also, data needs to be modeled using an appropriate alternative to make the required analysis possible. The life cycle of big data analysis using graph models (other models are possible, but

[2] Dashboard: https://itlab.uta.edu/mlndash-live/.

not addressed in this paper) is shown in Fig. 2. The dashboard detailed in this paper includes all the components of the life cycle shown in Fig. 2. Configuration files are used to generate graphs or MLN layers. Configuration files are also used for MLN analysis. Drill-down information is maintained and used along with visualization for easy understanding of results.

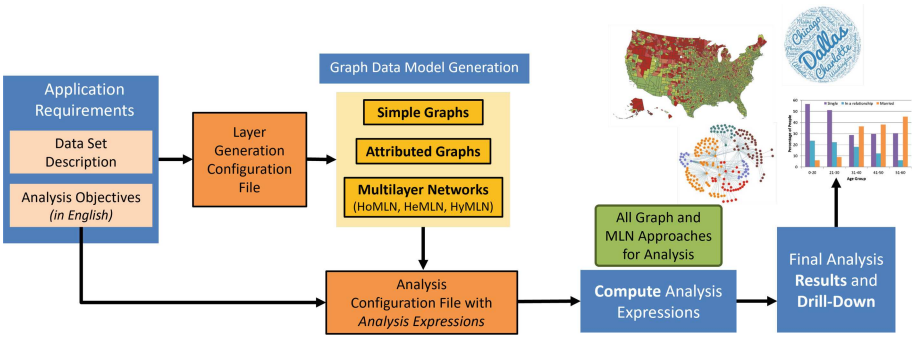


Fig. 2. Complex Data Analysis Life Cycle that need to be Supported

2.1 Space and Variety of MLN Algorithms

Several types of algorithms can be developed for MLN analysis ranging from reducing an MLN to a simple graph to use existing algorithms to developing **new** algorithms using the entire MLN, Between these two, there is another option that can leverage the first approach without developing **totally new** algorithms. They are briefly described below to show the space of algorithms that need to be supported by the interactive tool.

1. Traditional Aggregation Approach: In this approach (shown in Fig. 3(a)), layers of an MLN are conflated into a simple graph using type-independent or projection-based approaches. The aggregation process can be costly (depending on the number of layers) and the resulting graph can be large (for OR composition). The conversion, in either case, loses information [19,

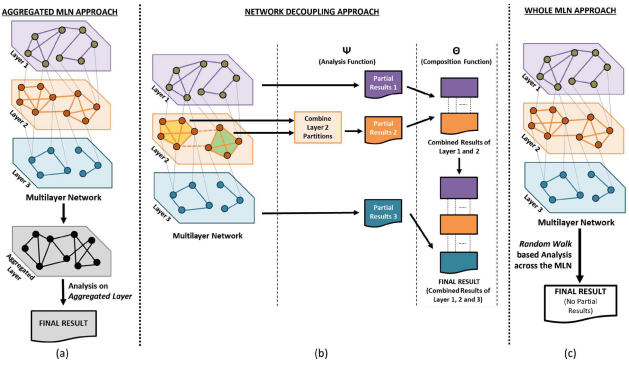


Fig. 3. Traditional vs. Decoupling vs. Holistic Approaches

22] for nodes, edges, labels, and further their relationships are not differentiated[3]. The advantage of this alternative is that extant algorithms (e.g., Louvain, Infomap, Degree Centrality, Substructure Discovery, etc.) can be used for analysis. This approach is not amenable to parallel processing as each layer cannot be processed independently. The loss in MLN structure, due to aggregation, also makes it difficult to drill-down results without maintaining extensive mapping. Finally, for processing a subset of layers, separate aggregation is required – making the approach less flexible and inefficient.

2. Decoupling-Based Approach: It is a "divide and conquer" approach (shown in Fig. 3(b)) for analyzing multilayer networks. The goal is to find the desired metric in each layer independently and then use a *separate* **composition function** to combine the results to generate what is missing for that analysis (e.g., community, centrality, substructure, motif, etc.) The *network decoupling* approach has been established as one of the better ways for analyzing MLNs in terms of efficiency and flexibility [34–36].

3. Holistic Approach: In this approach (shown in Fig. 3(c)), neither the MLN is aggregated nor the decoupling approach is used. An algorithm is developed from scratch (as available graph analysis algorithms cannot be used as in the previous two approaches), keeping the structure and semantics intact. However, a new algorithm needs to be developed for each analysis making this approach complex as the algorithm needs to traverse back and forth across layers. This approach has been used for coherent clustering in [11].

Given the number of approaches and the many metrics used for analysis, making these available in one place (along with descriptions, advantages, disadvantages, and even examples) is critical for data analyzers from non-computer science domains. This is a clear justification for not only the tool, but its modularity and extensibility.

2.2 Sample MLN Algorithms to be Supported

Table 1 shows a subset of analysis algorithms (for community, centrality, ...) available for graphs and MLNs that need to be part of any tool that supports the complex data analysis life cycle.

Currently, the dashboard described in this paper includes a large subset of the available sample algorithms shown in Table 1. This and the workflow shown in Fig. 2 have allowed us to test the modularity and extensibility of the approach described in this paper.

3 Related Work

Graphical interfaces and visualizations are not new and there exists a wide variety of tools for visualizing both base data, results, and drilled-down information in multiple ways [1,4,6,12]. Our focus, in this paper, is to make use of available visualization tools in the best way possible and not propose new ones. For example, we have experimented

[3] If one does not want to lose information, extensive mappings are needed for converting the MLN and its analysis results back to MLN.

Table 1. Sample set of Graph and MLN Analysis algorithms. Only a sample set is included due to space constraints

Analysis Type	Graph Type	Algorithm Description
Community Detection	Simple Graphs/ MLN Layers	`Louvain` [10], `Infomap` [12], `Walktrap` [32], `Fast-Greedy` [15], `Multilevel` [10], `Leading-Eigenvector` [28]: Simple graph community detection algorithms
	HoMLN	`CE-AND`: Intersection of Community Edges [34,36]
		`CE-OR`: Community detection on an OR-meta graph with AND-communities as meta nodes [36]
		`MiMAG`: Coherent subgraph (Community detection) for the whole MLN [11]
	HeMLN	`Maximum Weight Matching (MWM), Maximum Weight Matching with Ties (MWMT), Maximum Weight Perfect Matching (MWPM), Maximum Weight Relaxed Matching (MWRM)`: Matching algorithm for bipartite meta graph with communities as nodes and edge weights as the number of inter-community edges (ω_e), hub participation (ω_h) or density (ω_d) [38]
Degree Centrality	HoMLN	`DC-AND`: Estimated high degree nodes across 2 layers [35]
		`DC-OR`: Estimated nodes with high degree in at least 1 layer [30]
	HeMLN	`HeMLN-DC-AG`: Accuracy Guarantee Heuristic [26]
		`HeMLN-DC-PG`: Precision Guarantee Heuristic [26]
Betweenness Centrality	HeMLN	`HeMLN-BC`: Estimated high betweenness MLN nodes [25]
Closeness Centrality	HoMLN	`CC-AND`: Estimated high closeness nodes across 2 layers [31]
Substructure Discovery	Simple Graphs/ MLN Layers	`SUBDUE`, `MLN-SUBDUE`: Map/Reduce Algorithm for substructure discovery in large graphs [13,16,40]

with a wide variety of tools including, maps, individual graph and community visualization, animation of features in different ways, hovering to highlight data, and real-time data fetching and display, based on user input from a menu. The main contribution is the life cycle approach we have taken for analyzing complex data using graph and MLN models. To the best of our knowledge, we have not come across a dashboard that serves all aspects of MLN usage including modeling, generation of layers, analysis of generated layers as well as drill down & visualization. In addition, our architecture uses a common back-end to drive different user interaction and visualization front-ends. We have also paid attention to efficiency at the back-end by caching previously generated results with an efficient data structure for lookup.

Existing dashboards (Py3Plex [41], MultiNetx [27], Muxviz [18], and Pymnet [23]) are limited in their ability to analyze MLNs and mainly focus on visualizing the results as graphs. Some dashboards focus on reporting and visualizing raw data using maps [1,7,8] or time series plots and statistical modeling [4,5]. In contrast, this dashboard is about complex data analysis in the way the end user wants. That is, the end user should be able to control graphs or layers generated, the kind of analysis to be performed

once the graphs/layers are generated, and further visualize the results (or even base data) in multiple ways. Drill-down is integrated with visualization to make the model representation efficient for analysis. For example, minimal information is used for the model and mapping of label information is **generated** to be used for drill-down as part of visualization. A number of existing simple graph and MLN algorithms have been incorporated. The decoupling-based ones make it easier to perform drill-down using only layer information for recreating the structure [42]. The schema generation also separates information needed for drill-down (Relations) and information needed for analysis (MLNs) from the same Enhanced Entity Relationship (EER) diagram [37].

As this dashboard focuses on facilitating each user to perform the analysis appropriate for application data in their domain and hide the complexity of algorithms and their understanding, a flexible configuration file strategy is used for specifications (along with examples.) Layer generation can be done using multiple metrics and are easy to specify. Similarly, MLN analysis, whether simple or complex, can be specified using known operators (e.g., community, substructure, centrality, etc.) and precedence for evaluation. Input and output formats are transparently handled by the dashboard. Multiple visualizations can be generated and compared side-by-side and the ability to create files and directories for proper organization and management of data and results is provided.

4 Challenges to be Addressed

A graphical tool that caters to end users (who are likely to be domain experts, but may not be experts in the details of data analysis algorithms) by providing support for all stages of complex data analysis (i.e., life cycle) requires an easy-to-use interface that aids in the clear understanding of results of each step that can be stored for future lookup or sharing. The overall requirement can be succinctly expressed as: i) interface should be intuitively navigable, ii) minimize the number of clicks used to get a job done, iii) pro-actively prompt for available actions at any point (to prevent mistakes), and iv) provide minimal but useful meta information. These principles were used not only for the front-end, but also for individual module usage. The choice of tech stack and packages used for each module are discussed in Sect. 6.

The broader steps of data analysis involve: i) loading the data in the *specified* format (for the time being which will be extended to other formats), ii) generating appropriate layers of MLN required for analysis, iii) expressing and performing the analysis, and iv) visualizing base data and/or results to understand them. Data is assumed to be formatted as a csv (comma separated values) file which is widely used in mining and data analysis applications. Loading of data is done by choosing a local file and uploading it into a specific destination folder within the user space of the dashboard. Dashboard provides a navigable user directory structure (similar to what users are familiar with) as created by the user. Below, we discuss the main challenges for each of the back-end modules.

Modeling or Layer/Graph Generation: The primary challenge for the creation of layers is to choose a metric for generating layer graphs. It should support widely used metrics as well as allow users to customize their metrics as needed. The metric needs to be specified on the input data including the generation of inter-layer edges for HeMLNs.

In addition to the columns used for a node (it can be more than one column as in lat and long), a computation metric (such as distance) and a threshold (such as distance being less than the threshold) need to be specified. In addition to supporting widely used similarity metrics (such as equality, Haversine, Jaccard etc.), we want this to be customizable as well. Different feature types, such as nominal, categorical, numeric, date also need to be supported. Generation of inter-layer edges, on the other hand, requires two nodes that are likely to be from two different files along with a join condition for determining whether the nodes are connected. Apart from this, input file locations, the output locations need to be specified unambiguously. Details of how this is accomplished are described in Sect. 6.1.

Graph and MLN Analysis: Once the appropriate layers or graphs are generated and automatically stored in the directory specified, many kinds of analysis can be performed. Again, different analyses can be performed on the same generated layers. For example, if we generate 3 separate layers for the social networking applications – Facebook, LinkedIn, and X (formerly, Twitter), these layers can be analyzed individually or in any combination. These need to be specified in some way to compute and store the results. Multiple algorithms may exist for analysis. For example, there are several community detection algorithms (Louvain and Infomap being more popular) and it is the user's choice as to what is best for their analysis. Note that each algorithm may generate different communities (based on the approach and heuristics used). Multiple analyses as well as different combined analyses can be performed which needs to be expressed clearly. We use the infix form for expressing analysis expressions with precedence and each analysis expression is computed and its result stored as specified. An infix to post-fix conversion is done prior to evaluating the given expression. Optimization is incorporated by storing analysis sub-expression results to be re-used in later expressions by the same user. Details are in Sect. 6.2.

Drill-Down and Visualization: Analyzing results of complex analysis resulting in large output is extremely difficult and is further exacerbated by the fact that the result is in some encoded form without the semantics (e.g., labels) to understand it. Hence, a visualization that includes the original labels (drill-down) is more appropriate. Even when it is visualized, some visualizations may be more useful for understanding compared to others. For example, to identify an accident-prone region, visually, a word cloud representation may be better than looking at the dense graph (unless it is plotted on a map). Hence, it is imperative that in addition to drill down, alternate visualizations be provided. Another important issue is the size of results and scalability of visualization. This is a hard problem and there is ongoing research to address this. Interactive pan and zoom features can help deal with size. Rendering response time is also a consideration.

In this dashboard, we support drill-down transparently keeping mapping information during layer generation. This is superimposed on the visualization to show relevant attributes as part of visualization. We have also chosen different visualization alternatives, such as word cloud, Bokeh, interactive graphs, etc. to provide meaningful alternatives based on what the user is looking at. They can also be visualized side-by-side in different windows for better understanding. More details are provided in Sect. 6.3.

Finally, designing each module to work with other modules and sharing minimal information is a challenge in itself. Information is shared by persisting relevant data using efficient data structures. This will allow independent and parallel development of modules rather than waiting on each other. As each module needs slightly different skill sets, it will also make it easier for different groups to work on different modules by robustly sharing data. This will allow the dashboard to automatically detect and enforce certain dependencies. For example, if an analysis is specified on a layer that has not been generated, it will be detected and notified to the user including what needs to be done. This is critical as actions can be taken independently, and the user may not be aware of the inherent dependencies.

5 Modular and Extensible MLN-GeeWhiz Architecture

As shown in Fig. 4, front-end only interacts with the user and displays results. The back-end has several modules including registration and login. A lightweight and efficient relational database (SQLite) is used to store user meta information (email, password, etc.)

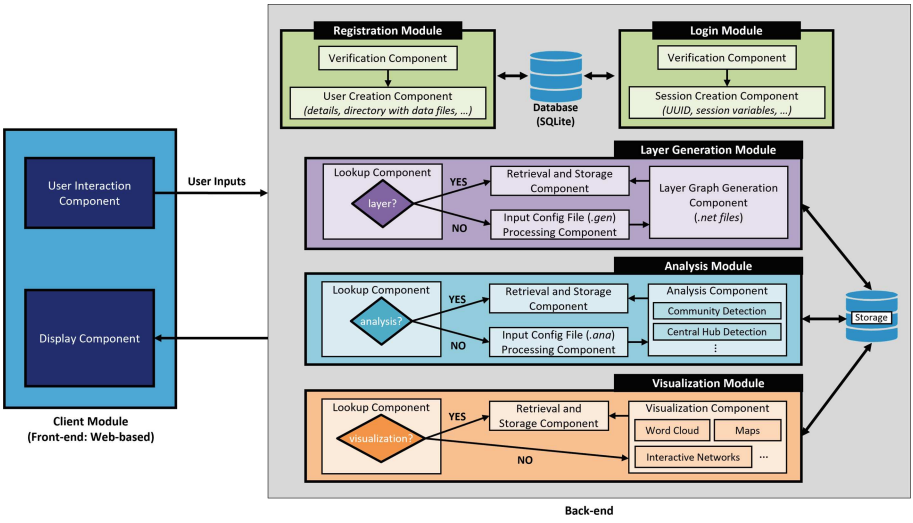


Fig. 4. MLN-geeWhiz Dashboard Architecture

Upon user registration, each user is allocated his/her own directory space. This ensures the isolation of user data for privacy and selective sharing as desired by the user. Each user starts with a pre-specified directory structure that can be extended. All files generated by the dashboard are also stored in the specified user directory.

The dashboard prompts actions based on the file selected and its extension. For example, the dashboard allows the generation of layer files by utilizing the "Generate Layers" button only when a configuration file with .gen extension is selected. The

dashboard pro-actively shows the possible actions based on the file chosen. This is further extended to choose the visualization option on files that can be visualized. The log data created by any module is shown in the display panel where information regarding the files generated, such as file name, destination folder, generation time (in seconds), number of nodes, number of edges, and number of connected components is shown.

The back-end includes major modules for analysis: layer generation, graph and MLN analysis, and drill-down & visualization. Each is separate and is implemented using different skill sets and in parallel. The flow in each module is kept as similar as possible. Communication between modules is through persistent data created and used by all modules. For example, Python dictionary is used to store layer names generated along with additional information (e.g., timestamp) and is accessed by other modules through the import command.

We use Flask [2] as the core framework for our application. Flask handles user authentication, registration, and routing requests to the appropriate modules based on user interactions. This seamless integration enables smooth data flow and interaction between different components of our application. HTML and CSS are used for front-end development, providing an intuitive and user-friendly interface for interacting with our application. Our front-end components are designed to facilitate easy navigation and interaction with the data analysis functionalities offered by our application. The control flow of our application is managed by Flask, which routes requests from users to the appropriate modules based on their actions. As an example, when a user initiates a data analysis task, Flask routes the request to the analysis module, which generates the necessary files and returns control back to the Flask application upon completion.

5.1 Technology Comparison and Selection Rationale

To ensure that our application meets the high standards required for performance, privacy, security, and user experience, we evaluated different technologies to finalize the tech stack. Table 2 shows a comparison of technologies considered, along with the reasons for our final selections.

Table 2. Comparison of Technologies considered for **`MLN-geeWhiz`** Architecture

Category	Technology Chosen	Considered Options	Selection Rationale
Front-end	**HTML/CSS**	Angular, React, Vue.js	HTML/CSS has been chosen for its simplicity and broad support. It allows for rapid prototyping and is sufficient for our current UI needs without the overhead of a more complex framework
Back-end	**Flask**	Django, Express.js, Ruby on Rails	Flask's lightweight and modular nature makes it ideal for our application, offering flexibility and ease of development for our RESTful API design
Database	**SQLite**	PostgreSQL, MySQL, MongoDB	SQLite DB provides a lightweight, file-based solution, eliminating the need for a separate database server, which suits the scale of our current user base and data

5.2 Session Management

Session management is important when multiple users are using the dashboard. It needs to be thread-safe to make sure session data is isolated and routed properly. User-specific data, including file paths and user identifiers need to be *isolated* and not shared with other users. For this purpose, we use the session management capabilities provided by Flask. Flask offers a secure and scalable way to manage user sessions by assigning a unique session ID to each user upon login. This session ID is then used to track and store user-specific data in a way that is segregated from other users' data.

The implementation of dedicated Flask sessions results in a secure and reliable user experience. Each user's session is properly isolated, ensuring that users can only access their own files, even when multiple users are logged in simultaneously. This not only takes care of the security issues but also enhances the overall integrity and trustworthiness of our application.

6 Back-End Modules

In this section, we will summarize the functionality of each module, module architecture, and interaction with other modules and the front-end.

6.1 MLN Layer Generation

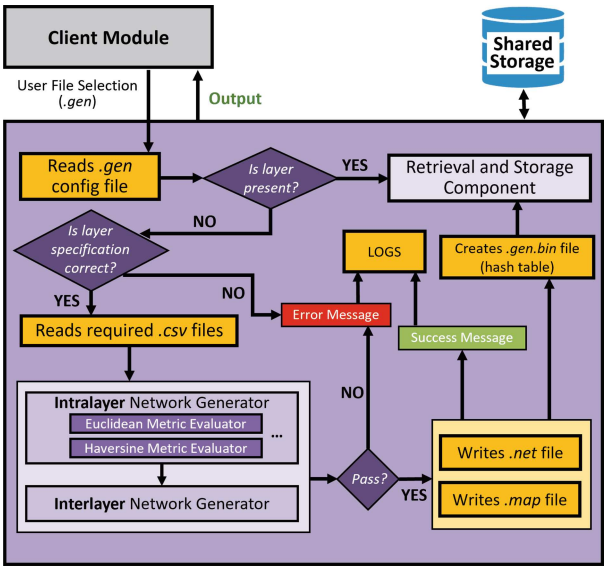


Fig. 5. Layer Generation Module Flow

Developing the layer generation tool for the multilayer network entailed users to specify precisely how nodes and edges of layers/graphs are generated using input data files. In addition to widely used metrics (given later in this section), we want to support custom metrics that can be provided as Python code. Currently, every line in the input data file (csv) becomes a node in the layer. For edges, one issue involves the use of multiple attributes for nodes where each combination needs to have non-null values for a specific set of node attributes. The node attribute combinations for layer generation are designed to allow users to choose the appropriate one based on their needs. We devise a method to

enable users to generate multiple layers utilizing a single configuration file. The configuration file (.gen extension) contains information regarding the input directory, output directory, username, and the property specifications used for each layer generation. Both intra-layer and inter-layer can be generated using the same input configuration file. Each layer generation is specified separately. Specification for the intra-layer generation differs from that of inter-layer, allowing for two distinct types of layer generations with a single configuration file. Figure 5 shows the workflow of the layer generator module. Appropriate error codes are sent to the client module which displays messages for recovering from the error. Successful generation of layers for a configuration file is indicated.

Intra-Layer Edge Generation: Several widely-used metrics are supported: equality, Jaccard, Euclidean, Haversine, cosine, range, multi-range, fixed range for determining whether an edge exists between two nodes. An edge between two node IDs is established using a similarity metric which compares the similarity between two node IDs using a comparison of column attribute(s). The value returned by the comparator is checked against the threshold value specified by the user. If the value returned is below the threshold value given, an edge is established between two node IDs. The following similarity metrics are supported:

- **EQUALITY**: This metric checks whether two values are equal or not.
- **EUCLIDEAN**: The Euclidean distance between the two values, a and b, is calculated by taking the absolute value of their difference, $|a - b|$.
- **HAVERSINE**: The haversine distance is calculated by formula $d = R * c$, Here R is earth's radius (mean radius = 6,371km) and $c = 2 * a * tan2(\sqrt{a}, \sqrt{1-a})$ and $a = sin^2(\frac{lat}{2}) + cos(lat1) * cos(lat2) * sin^2(\frac{long}{2})$ where lat=lat2-lat1, long=long2-long1 (lat and long represents latitude and longitude respectively).
- **JACCARD**: The similarity between two sets A and B is measured using the equation: $J(A, B) = \frac{A \cap B}{A \cup B}$.
- **COSINE**: The cosine similarity between two vectors A and B is calculated using the following equation: $Cosine(A, B) = \frac{A * B}{||A|| ||B||}$ where $||\ A\ ||$ and $||\ B\ ||$ are the euclidean norm of vector A and B respectively.

Table 3. Different Types of Feature

Feature type	Description
NOMINAL	It represents a category or state. For example: political belief of a person can be **democratic**
NUMERIC	It represents integer or floating point. For example: speed of a vehicle in mph can be **75**
GEOGRAPHIC	It represents longitude feature column and latitude feature column. Both the columns contain numeric value
TIME	It represents the time in the format **HH-MM**
DATE	It represents the date in the format **DD-MM-YYYY** or **MM-DD-YYYY**
SET	A set with m members has the format: {value1, value2, ... valuem}. For example: genre of a movie is represented by {**ACTION, COMEDY, DRAMA**}
TEXT	It represents string of any length. For example: **Tweet**: **Congratulations, France FIFA 2018 Winners, IMDB Review**: **The movie was crap**

For similarity metrics - range, multi-range and fixed range, the generator verifies if, for a given pair of node IDs, the associated comparison column value falls within the specified interval for establishing the connection between two node IDs. For the multi-range, the user inputs numerous ranges. Two node IDs are connected if their comparison column value lies inside the same range. The user-provided parameter fixed-range is separated into equal-sized parts using the number of segments attribute. Two node IDs are connected if their comparison column values fall within the same segment. The user has the option to select either an open or closed interval for the range, and multi-range depending on their specific needs. Another challenge was to accommodate the tool to support using single or multiple columns as nodes IDs. For instance, a single column $Accident\_id$ can be used as node column. Again, two location columns ($Location\_Easting\_OSGR$, $Location\_Northing\_OSGR$) can be used as nodes IDs for layer generation. To solve this issue, the tool maps both single and multiple-column values to a single node ID. Mapping is necessary for analysis purposes like community detection. The tool also supports a variety of data types: nominal, numeric, geographic, date, time, set, text. Table 3 provides the description of different data types allowed by the layer generation tool for comparison column. Below is an example of a layer generation configuration, that connects two accident nodes which occurred within 50 km of each other based on haversine distance.

```
BEGIN_LAYER
INPUT_FILE_NAME= large-5KUKAccident-2014.csv
LAYER_NAME=distance_between_accidents
PRIMARY_KEY_COLUMN=Location_Easting_OSGR,Location_Northing_OSGR
LONGITUDE_FEATURE_COLUMN=Longitude
LATITUDE_FEATURE_COLUMN=Latitude
FEATURE_TYPE=NUMERIC
SIMILARITY_METRIC=HAVERSINE
THRESHOLD=50
END_LAYER
```

Inter-Layer Generation: Requires merging two node IDs based on a join column to depict a specific relation between the merged node pairs. It is done as follows. The layer generation tool needs to generate two layers before generating an inter-layer. Then, the tool creates an inter-layer file from the input csv files required to generate layers. The inter-layer is generated by linking the node IDs from both layers. The two node IDs from two csv files used for layer generation are connected by matching them based on a join attribute specified in the configuration file. The relation between two node IDs from two layers is depicted in the generated inter-layer using the relationship attribute from the configuration file. Below is an example of a configuration file with property specifications for the inter-layer generation between the Actor and Director layers. This will connect nodes from the actor and director layers who have worked in the same movie.

```
BEGIN_INTERLAYER
LAYER_1_NAME=AvgRating_Actors
LAYER_1_INPUT_FILE_NAME=AvgRating_Actors.csv
LAYER_2_NAME=AvgRating_Directors
LAYER_2_INPUT_FILE_NAME=AvgRating_Directors.csv
JOIN_COLUMN_NAME=Movie_ID
RELATIONSHIP_NAME=works_with
INTER_LAYER_NAME=actor_director_works_with
INTER_LAYER_GENERATION_TYPE=System_Generated
END_INTERLAYER
```

Although part of the MLN dashboard, the layer generation tool is an independent module. The generating tool accepts the user information and configuration file path as input parameters. The client module invokes the generation module by providing these two parameters to execute layer generation.

6.2 MLN Analysis Module

Although an independent module, the analysis module seamlessly integrates with the layer generation, directly utilizing the generated layers for analysis. This is done by persistent dictionaries written by each module and used by the other modules. Upon completion of the analysis, the results are readily available for the visualization module, where users can view and interpret the findings through various visualizations.

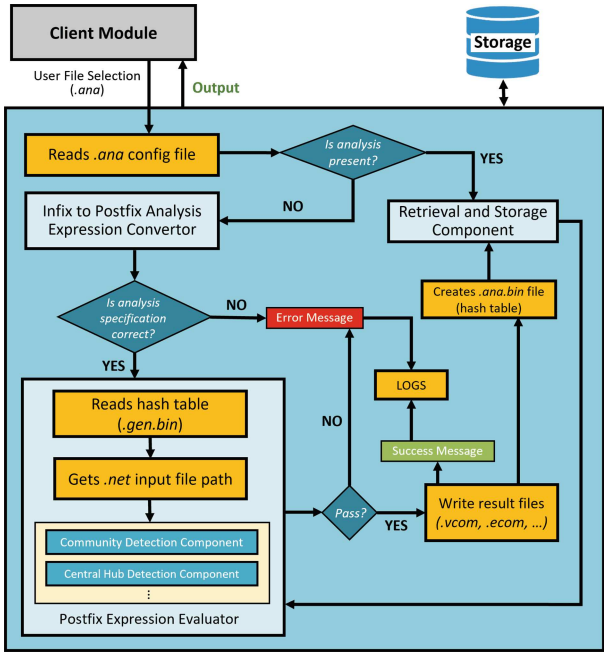


Fig. 6. Graph/MLN Analysis Module Flow

This ensures a smooth and efficient workflow from data input to result visualization. The analysis process is carried out through the utilization of an analysis configuration file (with extension.ana). This file serves as a blueprint, containing expressions that specify the analysis computations to be performed. An infix expression (with explicit precedence) is used for specifying the analysis and a name for storing the results. Workflow within the analysis module is shown in Fig. 6.

Analysis configuration file (.ana extension) indicates the layers to be used

by importing the layer generation configuration file name that was used for generating layers. This can be specified unambiguously and multiple configuration files can be imported to analyze layers generated at different times by different configuration files. The analysis expression is specified in the widely used infix form and is easy to understand. The operators are specific to graph/MLN analysis and the domain user knows the analysis being performed. A sample infix expression along with its translated post-fix expression is shown below.

Infix Analysis Expression: `Louvain(NOT layer1) CV-AND (Louvain (layer2 OR layer1) CE-AND Infomap(layer3 AND layer1))`

Postfix of the Above Expression: `layer1 NOT Louvain layer2 layer1 OR Louvain layer3 layer1 AND Infomap CE-AND CV-AND`

Observe the use of analysis operators (Louvain [10] and Infomap [12]), Boolean operators (AND, OR, NOT) [34], and multilayer composition operators CE-AND [36] and CV-AND [34]. In order to honor the precedence, an infix expression needs to be converted into its postfix form for evaluation. The operands here are files containing graphs or results of previous computations.

Analysis Optimization: Some of the algorithms used for graph analysis take a significant amount of time to compute depending on the number of nodes, edges, and other graph properties (e.g., degree). Also, observe that the same analysis is likely to be done on the same layer for two different (sub) expressions. Redoing the same analysis on the same layer using the same algorithm increases analysis time and hence response time.

To overcome the above, primitive layer computations are time-stamped and checked for their presence using the dictionary. If they have been generated earlier, they are not regenerated during the analysis. This has reduced the time for analysis significantly, especially when the graph sizes are large.

Use of Multiple Languages for Analysis: Many available simple graph analysis algorithms are in C++. Integrating C++ methods with the main analysis Python driver presented some challenges in communication and data handling between the two languages. We utilized C++ for computationally intensive analysis methods while leveraging Python for overall control and orchestration. A robust system was developed to facilitate seamless data exchange and function calls between the two languages.

6.3 Drill-Down and Visualization

Developing an effective drill-down and visualization is a challenge due to the complex nature of multilayer networks (MLNs) and the various requirements of users. Key challenges include:

– **Availability of alternatives**: We aim to provide users with multiple visualization options to cater to different analysis needs. Implementing various visualization techniques while ensuring user-friendly interaction is the key.

- **Handling Large and Complex Data Sets**: MLNs often involve large graphs from various domains, making it difficult to visualize them in the screen space available.
- **Integrating Interactivity**: Enabling interactive features such as pan, zoom, hover, and drill-down capabilities within the visualization interface required careful design and implementation to ensure a smooth user experience.

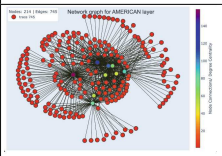
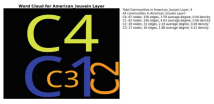
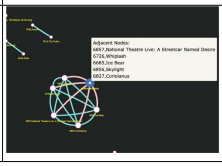
To handle large graphs meaningfully, we adopt a design where the options provided are based on the number of nodes and edges. To further enhance user engagement and exploration, the dashboard incorporates interactive visualization features. Users can dynamically interact with visualizations, such as zooming in on specific network regions, hovering over nodes for additional information, and drilling down into analysis results.

The Visualizer module relies on layer generation and analysis modules for input file types such as `.net` files (Network Files), `.ecom` files (Edge Community Files), and `.vcom` files (Vertex Community Files). Additionally, the module utilizes a primary mapping input file (`.map`) generated by the layer generation module to map labels or node attributes with respective node IDs in the visualization when hovered. The following types of visualizations are offered to facilitate comprehensive analysis:

1. **Network Visualization Type**: Utilizes the Plotly Python module to generate a static graph where node colors are determined based on the number of connections.
2. **Degree-Centrality Visualization Type**: Focuses on visualizing degree centrality.
3. **Community Visualization Type**: Provides insights into community structures within the network, designed for `.ecom` and `.vcom` input files.
4. **Word Cloud Visualization Type**: Specifically tailored for `.ecom` input files, generates word clouds representing node attributes and ledger information such as degree centrality and top 10 communities of the layer.
5. **Interactive Visualization Type**: Utilizing the PyVis Python module to offer an interactive experience where users can hover over nodes, toggle node visibility, and explore network structures in detail.
6. **Map Visualization Type**: Available for input mapper files containing respective longitude and latitude data, this visualization allows users to visualize node locations on the map, enhancing the understanding of the analyzed data.

Table 4 shows the packages used for different visualization needs and the rationale for choosing them.

Table 4. Comparison and rationale of Visualizations Used in MLN-geeWhiz

Packages Used	Reason for selection and Use Cases	Visualization Snippet
Plotly	Generates a static network graph with node colors to reflect the number of connections, enabling easy identification of network hubs and isolated nodes.	
Plotly (Map)	Employs Plotly's robust mapping tools to visually represent geographic data, making it excellent for highlighting relationships and patterns, such as accident-prone areas.	
Bokeh (Community Detection)	Provides interactive capabilities, making it suitable for visualizing community structures within networks.	
Bokeh (Degree Centrality)	Renders networks with nodes sized according to their degree centrality, providing a clear visual hierarchy that identifies and ranks key influencers or hubs.	
Wordcloud	Creates an engaging word cloud that summarizes key attributes of network communities, such as the density and average degree.	
PyVis	Seamlessly integrates with NetworkX to provide a fully interactive network visualization experience, where users can manipulate nodes and edges in real time to gain deeper insights into the network structure.	

7 Real-Word Data Analysis Using the MLN-GeeWhiz Dashboard

In this section, we will demonstrate the use of the dashboard for some applications to show its utility, functionality, and visualization options supported.

UK Accident Use Case: We have used the UK accident data set [9] for this. Some sample requirements for an accident data set are finding out the (i) accident-prone regions, (ii) days of the week and time periods that witness an increase in the number of accidents, (iii) the most unsafe lighting, weather, and road conditions, (iv) type of locations that become the epicenter of accidents, and so on. Such insights can help city planners to take safety precautions.

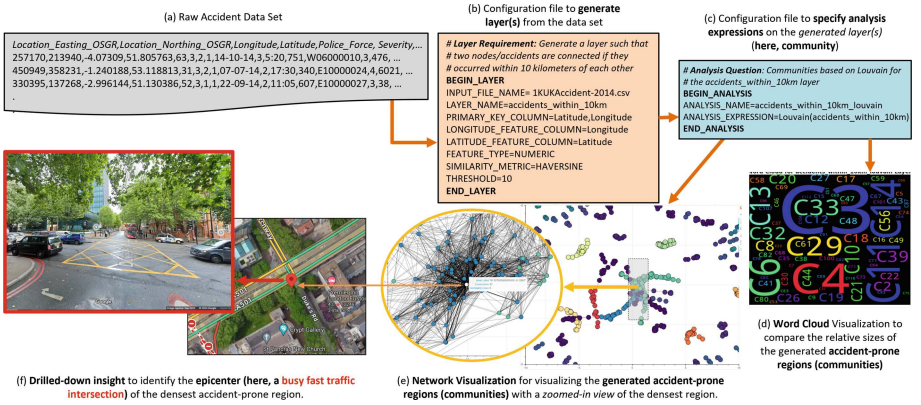


Fig. 7. Showcasing the use of the MLN-Dashboard to model, analyze, drill-down and visualize a raw accident data set.

Figure 7 shows the entire workflow from a raw accident data set to the final drill-down visualized analysis insight. Figure 7 (a) shows the snapshot of the UK Accident data set which for every accident occurrence captures the location (latitude, longitude), light conditions, weather conditions, date of occurrence, and so on.

The user-specified configuration file in Fig. 7(b) connects two accident nodes if they have occurred within 10 km of each other based on Haversine distance to generate a layer. Then, the configuration file in Fig. 7(c)[4] detects communities (i.e., in this case accident-prone regions) using the Louvain algorithm [10] with the output stored as a text file containing *(edge, communityID)* pairs. These accident-prone regions (communities) have been visualized using a word cloud to understand the relative severity of the regions in Fig. 7(d). Alternatively, network visualizations of the accident-prone regions with zoom-in, hover, and pan facility help us understand the connectivity within the *densest* communities (Fig. 7(e)). Finally, Fig. 7(f) gives us drilled-down insight to identify that the epicenter of the densest accident-prone region (i.e., node with the highest degree in the densest community) is a fast traffic intersection on the Google Maps using the latitude, longitude node label.

In addition to what is shown in this paper, the MLN-Dashboard has been used to analyze other raw data sets, such as IMDb [3] to find the groups of movies that are most similar to each other based on the genres or the highly rated actors who have never worked together; US Airline data sets have been analyzed to identify airlines hubs or cities from which you can reach other cities with least number of hops or flight changes using closeness centrality analysis, and so on. The DBLP data set has been used to discover author groups that have published significantly in some conferences over 20 years, authors who publish in top-tier conferences, but not in other conferences, etc.

[4] A complete example of the layer generation configuration file is shown in Sect. 6.1.

8 Conclusions

In this paper, we have presented an interactive tool architecture for supporting complex data analysis using graph models including multilayer networks. The dashboard is conceived for the whole life cycle and designed to cover all stages of life cycle in an easy-to-use, pro-active manner. Special attention has been paid to prevent user mistakes by prompting what is allowed. For example, only permitted actions are highlighted at each step. The architecture and modularity, based on functionality, provide *flexibility* (of development and optimization), *extensibility* (of visualizations, analysis, layer and inter-layer graph generation, and data set formats), *consistency for multiple concurrent users who may share data and results*, and *efficiency* (for response time by using previously computed results using timestamp checks).

Future work includes extending the dashboard into a full-fledged portal providing selective uploads and sharing (of data, algorithm code, as well as results), community interaction, and comparison of algorithms and creation of benchmarks while taking care of challenges like security, load balancing, performance issues and so on.

Acknowledgments. This work was partially supported by NSF awards #1955798 and #1916084.

References

1. The centre for disease control covid dashboard. https://covid.cdc.gov/covid-data-tracker/
2. Flask web framework. https://palletsprojects.com/p/flask/
3. The internet movie database. ftp://ftp.fu-berlin.de/pub/misc/movies/database/
4. Johns hopkins university covid dashboard. https://www.arcgis.com/apps/opsdashboard/index.html#/bda7594740fd40299423467b48e9ecf6
5. The new york times covid dashboard. https://www.nytimes.com/interactive/2020/us/coronavirus-us-cases.html
6. The university of washington covid dashboard. https://hgis.uw.edu/virus/
7. The world health organization covid dashboard. https://covid19.who.int/
8. Worldometer covid statistics. https://www.worldometers.info/coronavirus/country/us/
9. Road safety - accidents 2014 (2014). https://data.gov.uk/dataset/road-accidents-safety-data/resource/1ae84544-6b06-425d-ad62-c85716a80022
10. Blondel, V.D., Guillaume, J., Lambiotte, R., Lefebvre, E.: Fast unfolding of community hierarchies in large networks. CoRR **abs/0803.0476** (2008). http://arxiv.org/abs/0803.0476
11. Boden, B., Günnemann, S., Hoffmann, H., Seidl, T.: Mining coherent subgraphs in multilayer graphs with edge labels. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1258–1266. KDD 2012, ACM (2012). https://doi.org/10.1145/2339530.2339726
12. Bohlin, L., Edler, D., Lancichinei, A., Rosvall, M.: Community detection and visualization of networks with the map equation framework (2014). http://www.mapequation.org/assets/publications/mapequationtutorial.pdf
13. Bolaj, K., Santra, A., Chakravarthy, S.: Substructure discovery in heterogeneous multilayer networks. In: IEEE International Conference on Knowledge Graph, ICKG 2024, Abu Dhabi, UAE, December 11-12, 2024. p. accepted. IEEE (2024)
14. Calvó-Armengol, A., Zenou, Y.: Social networks and crime decisions: the role of social structure in facilitating delinquent behavior. Int. Econ. Rev. **45**(3), 939–958 (2004)

15. Clauset, A., Newman, M.E., Moore, C.: Finding community structure in very large networks. Phys. Rev. E **70**(6), 066111 (2004)
16. Das, S., Chakravarthy, S.: Duplicate reduction in graph mining: approaches, analysis, and evaluation. IEEE Trans. Knowl. Data Eng. **30**(8), 1454–1466 (2018). https://doi.org/10.1109/TKDE.2018.2795003
17. De Domenico, M., Lancichinetti, A., Arenas, A., Rosvall, M.: Identifying modular flows on multilayer networks reveals highly overlapping organization in interconnected systems. Phys. Rev. X **5**(1), 011027 (2015)
18. De Domenico, M., Porter, M.A., Arenas, A.: Muxviz: a tool for multilayer analysis and visualization of networks. J. Complex Netw. cnu038 (2014)
19. De Domenico, M., Solé-Ribalta, A., Gómez, S., Arenas, A.: Navigability of interconnected networks under random failures. Proc. National Acad. Sciences (2014). https://doi.org/10.1073/pnas.1318469111, https://www.pnas.org/content/early/2014/05/21/1318469111
20. Gomez, M., Garcia, S., Rajtmajer, S., Grady, C., Mejia, A.: Fragility of a multilayer network of intranational supply chains. Appl. Netw. Sci. **5**(1) (2020). https://doi.org/10.1007/s41109-020-00310-1
21. Hopkins, A.L.: Network pharmacology: the next paradigm in drug discovery. Nat. Chem. Biol. **4**(11), 682 (2008)
22. Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J.P., Moreno, Y., Porter, M.A.: Multilayer networks. CoRR **abs/1309.7233** (2013). http://arxiv.org/abs/1309.7233
23. Kivelä, M.: Pymnet: free library for analysing multilayer networks. http://people.maths.ox.ac.uk/kivela/mln_library/
24. Lara-Martínez, P., Obregón-Quintana, B., Reyes-Manzano, C., López-Rodríguez, I., Guzmán-Vargas, L.: A multiplex analysis of phonological and orthographic networks. PLoS ONE **17**(9), e0274617 (2022)
25. Mukunda, K., Roy, A., Santra, A., Chakravarthy, S.: Stress centrality in heterogeneous multilayer networks: heuristics-based detection. In: Ninth IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2023, Athens, Greece, July 17-20. IEEE (2023)
26. Mukunda, K., Santra, A., Chakravarthy, S.: The challenge of finding degree centrality nodes in heterogeneous multilayer networks. In: Proceedings of the 31st Italian Symposium on Advanced Database Systems, SEBD 2023, Galzignano Terme, Italy, July 2-5, 2023. CEUR Workshop Proceedings (2023)
27. multiNetx github Page (2019). https://github.com/nkoub/multinetx/blob/master/README.md
28. Newman, M.E.: Finding community structure in networks using the eigenvectors of matrices. Phys. Rev. E–Stat. Nonlinear Soft Matter Phys. **74**(3), 036104 (2006)
29. Pagani, G.A., Aiello, M.: The power grid as a complex network: a survey. Phys. A **392**(11), 2688–2700 (2013)
30. Pavel, H., Roy, A., Santra, A., Chakravarthy, S.: Degree centrality definition, and its computation for homogeneous multilayer networks using heuristics-based algorithms. In: Knowledge Discovery, Knowledge Engineering and Knowledge Management - 14th International Joint Conference, IC3K 2022, Valletta, Malta, October 24-26, 2022, Revised Selected Papers. p. Accepted. Communications in Computer and Information Science, Springer (2023)
31. Pavel, H.R., Roy, A., Santra, A., Chakravarthy, S.: Closeness centrality detection in homogeneous multilayer networks. In: Proceedings of the 15th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2023, KDIR, Rome, Italy, November 13-15, 2023 (2023)
32. Pons, P., Latapy, M.: Computing communities in large networks using random walks. J. Graph Algorithms Appl. **10**(2), 191–218 (2006)

33. Sahneh, F.D., Scoglio, C.: Competitive epidemic spreading over arbitrary multilayer networks. Phys. Rev. E **89**(6), 062817 (2014)
34. Santra, A., Bhowmick, S., Chakravarthy, S.: Efficient community re-creation in multilayer networks using boolean operations. In: International Conference on Computational Science, pp. 58–67. Zurich, Switzerland (2017)
35. Santra, A., Bhowmick, S., Chakravarthy, S.: Hubify: efficient estimation of central entities across multiplex layer compositions. In: IEEE ICDM Workshops (2017)
36. Santra, A., Bhowmick, S., Irany, F.A., Madduri, K., Chakravarthy, S.: Efficient community detection in multilayer networks using boolean compositions. Front. Big Data (2023)
37. Santra, A., Komar, K., Bhowmick, S., Chakravarthy, S.: From base data to knowledge discovery – a life cycle approach – using multilayer networks. Data Knowl. Eng. 102058 (2022). https://doi.org/10.1016/j.datak.2022.102058, https://www.sciencedirect.com/science/article/pii/S0169023X22000556
38. Santra, A., Komar, K.S., Bhowmick, S., Chakravarthy, S.: A new community definition for multilayer networks and a novel approach for its efficient computation. arXiv preprint arXiv:2004.09625 (2020)
39. Scabini, L.F., Ribas, L.C., Neiva, M.B., Junior, A.G., Farfán, A.J., Bruno, O.M.: Social interaction layers in complex networks for the dynamical epidemic modeling of COVID-19 in Brazil. Physica A: Stat. Mech. Appl. **564**, 125498 (2021)
40. Singh, A., Santra, A., Chakravarthy, S.: Scalable substructure discovery algorithm for homogeneous multilayer networks. In: 2024 IEEE International Conference on Big Data (BigData). IEEE (2024)
41. Skrlj, B., Kralj, J., Lavrac, N.: Py3plex toolkit for visualization and analysis of multilayer networks. Appl. Netw. Sci. **4**(1), 94:1–94:24 (2019)
42. Vu, X.S., Santra, A., Chakravarthy, S., Jiang, L.: Generic multilayer network data analysis with the fusion of content and structure. In: CICLing 2019, La Rochelle, France (2019)
43. Wang, Z., Wang, L., Szolnoki, A., Perc, M.: Evolutionary games on multilayer networks: a colloquium. Eur. Phys. J. B **88**(5), 124 (2015)