A Comparative Analysis of Multilayer Network Software

Christian Ocklind
Abstract

As the study of multilayer networks has increased in popularity in the recent decades, so has the number of available multilayer network software. However, few studies have compared this software in terms of scalability performance, overall features provided, and their underlying implementations and crucial design decisions. In this thesis, a selection of four multilayer network libraries: Pymnet, Multinet, multiNetX and muxViz were compared thoroughly. This was done by using citation network data from a publicly available dataset and creating a temporal multilayer network model from it. The libraries executed designed queries on this model for finding suitable operations to test their scalability. These operations were aggregation and degree calculation and the implementation of this whole analysis was written in Python and R. Pymnet and Multinet were applicable in all use cases, but muxViz performed the best when the scalability of all software was tested for the aggregation operation. When testing the scalability of calculating the degree of the nodes in a temporal network, weaknesses in the software were discovered. This work can be extended by testing additional multilayer network software and using several different datasets to construct different multilayer networks that are not restricted to one dimension.
Contents

1 Introduction .............................. 1
1.1 Purpose ................................ 1
1.2 Goal .................................. 1
1.3 Delimitations ............................ 2
1.4 Structure of the report ...................... 2

2 Background and prior work ....................... 3
2.1 Graphs ................................ 3
2.2 Multilayer networks .......................... 3
2.3 Temporal networks ........................... 5
2.4 Multiplex networks ......................... 7
2.5 Supra-adjacency matrices ..................... 8
2.6 Aggregation .............................. 9
2.7 Degree measurements and neighbourhood ...... 10
2.8 Layer difference .......................... 10

3 Methodology ................................ 11
3.1 Dataset overview .......................... 11
  3.1.1 Data format .......................... 11
3.2 Multilayer network model ....................... 13
  3.2.1 Converting the dataset to multilayer network format ... 14

4 Multilayer network software ....................... 17
4.1 Pymnet ................................ 18
4.2 Multinet ................................ 19
4.3 multiNetX ............................... 19
4.4 muxViz ................................ 20

5 Comparative analysis .......................... 22
5.1 Test environment ........................... 22
5.2 Tools used ............................... 22
5.3 Queries and data exploration .................... 23
  5.3.1 Query 1: Longest active journals .................. 23
  5.3.2 Query 2: Layer difference ..................... 24
  5.3.3 Query 3: Influencing journals .................. 25

6 Result .................................. 27
6.1 Query execution .......................... 27
6.2 Scalability testing ........................... 29

7 Discussion ................................ 31
7.1 Anomaly investigation ......................... 31
7.2 Potential overhead in Multinet ................... 31
7.3 Exploring coercion issues in muxViz ................ 31
7.4 Case Study: multiNetX memory consumption ........... 32
List of Figures

1. Two multilayer networks with different dimensions .......... 4
2. Temporal network as a sequence of undirected graphs ...... 5
3. Temporal multilayer network .................................................. 6
4. Multiplex network ................................................................. 7
5. A multilayer network and its supra-adjacency matrix ...... 8
6. Aggregating a multilayer network ................................. 9
7. Layer difference example ..................................................... 10
8. A simplified version of creating a directed graph from the raw dataset format .................................................. 12
9. Overview of multilayer network model used in method .......... 13
10. Simplified example of multilayer network data format ......... 14
11. Overview of how a node ID/label is mapped to its journal title using a data frame .................................................. 15
12. Subset of the dataset visualised in Pymnet ................. 16
13. Overview of computing the longest active journals ........ 23
14. Monoplex network with different node sizes and colours representing in-degree, for data exploration ..................... 25
15. Query result, longest active journals between 1900-1970s .... 27
16. Query result, top 20 influencing journals between 1900-2000 .. 28
17. Memory consumption over time: multiNetX degree calculation 1900-1960s .................................................. 33
18. Memory consumption over time: Multinet degree calculation 1900-1960s .................................................. 33
19. Memory consumption over time: Pymnet degree calculation 1900-1960s .................................................. 34

List of Tables

1. Feature comparison table of multilayer network software .... 17
2. Scalability testing results: Aggregation ......................... 29
3. Scalability testing results: Degree calculation ............... 30
1 Introduction

Networks provide a structure for establishing connections between entities and analysing their interrelationships. However, with the increasing complexity of networks being studied, traditional methods within network theory for modelling and analysing today’s systems may not always be sufficient compared to other alternatives that are accessible today. One alternative that has gained more attention in recent decades is multilayer networks. Simply put, the difference between a traditional network (also known as a monoplex network) and a multilayer network is that multilayer networks can represent additional types of properties, interactions and other characteristics in a structured manner using several “layers” [1]. For example, a past study from 2015 used multilayer networks to model the entirety of the United Kingdom’s public transport system for one week in 2010 [2]. This study used a multilayer network software implemented in R called muxViz to visualise the large amounts of data being represented [3]. As of today, there are several multilayer network software that can be used to construct, visualise and analyse multilayer networks in a similar way. For example, Pymnet, umnet.multinet (Multinet) and multiNetX are all Python libraries that are capable of performing similar studies when dealing with large amounts of data representing a complex system [4, 5, 6]. Although all these four software are meant for multilayer network analysis, they have been implemented with different approaches regarding which mathematical multilayer network frameworks they are based on and what types of multilayer networks are focused on for analysis.

This thesis will undergo a comparative analysis using four multilayer network software: Pymnet, Multinet, multiNetX, and muxViz. The analysis compares their performance, implementation, design, and how they can be used in a specific multilayer network analysis use case.

1.1 Purpose

Comparing various multilayer network software in a specific use case, including their design and performance differences, has yet to be extensively discussed. Therefore, this subject will be explored as multilayer network analysis is becoming increasingly popular and applicable in multiple fields.

1.2 Goal

This thesis aims to leverage the results from the comparative analysis to offer well-informed recommendations and suggestions on the most appropriate multilayer network software to use. For example, one of the software might be more efficient at completing a specific operation, whereas another is more flexible for general use cases when exploring multilayer network data. On the other hand, this study will also highlight any comparisons that might not have been fair tests during the experiments, as the software varies greatly in some aspects.
1.3 Delimitations

In order to solely focus on the comparison of multilayer network software, a publicly available dataset will be used for all of the software during the comparative analysis. Thus no data scraping or collection techniques are covered in this report. Furthermore, finding valuable information from the dataset will be attempted using the libraries. However, the validity of the results will not be prioritised over other comparison factors, such as testing the scalability of the software.

1.4 Structure of the report

To begin with, a chosen formal mathematical multilayer network framework from past literature is introduced to explain relevant terminology and definitions within the field. Once the chosen framework has been introduced, the chosen dataset and relevant data formats are presented. Next, an in-depth analysis of all of the software is conducted, where a feature comparison table is also introduced, summarising their difference in critical features for this thesis. The comparative analysis method is then introduced and conducted on all software. Finally, the results are presented, along with any anomalies and other insights from the study.
2 Background and prior work

In order to properly understand how the multilayer network software will be compared, it is necessary to clarify what concepts and definitions from past literature will be used for both networks and multilayer networks. Furthermore, this section will cover common types of multilayer networks and simplification methods that can be applied to general multilayer networks.

2.1 Graphs

Graphs are commonly used for modelling monoplex networks. A graph \( G = (V, E) \), where \( V \) is the set of vertices and \( E \) is the set of edges \[7\]. Therefore, the set of edges that connect two vertices is \( E \subseteq V \times V \). In other words, \( E \) is a subset of the Cartesian product of \( V \) and \( V \). If an edge connects two vertices, then these two vertices are adjacent to each other, and the edge is incident to these two vertices. Alternatively, a graph can be represented by an adjacency matrix consisting of \( |V| \) rows and \( |V| \) columns (a square matrix) showing which vertices are adjacent to each other in the graph \[7\]. Similarly, graphs can also be represented using adjacency lists where each list describes the adjacent vertices for a particular vertex in the graph. Edges can either be directed or undirected; if directed, the vertices connected by the edge have a one-way relationship. However, if the graph consists of undirected edges, the two vertices connected by an edge have a two-way/symmetrical relationship. The edges of directed and undirected graphs may also store weights in the edges, forming a weighted graph.

2.2 Multilayer networks

As the study of multilayer networks has accelerated in different fields, there has been a lack of consistency regarding terminology and a general mathematical framework for further analysis. Given this information, this thesis will follow the general framework for studying multilayer networks provided by Kivelä et al. \[4\], which attempts to consolidate the various terminology and frameworks used in past literature into one general framework.

A multilayer network is a quadruple \( M = (V_M, E_M, V, L) \), where \( V \) is the set of vertices and \( L \) is the set of layers in the multilayer network \[4\]. The set \( L \) includes all combinations of elementary layers, which is equivalent to \( L_1 \times \ldots \times L_d \), where \( d \) is the number of dimensions. The number of dimensions is determined by the cardinality of the set of elementary layers, denoted as \( L_1, ..., L_d \). Each set of elementary layers specifies the layers that belong to a particular dimension.

In the general framework, dimensions are referred to as ‘aspects’ to avoid confusion between the mathematical definition of a dimension. In this thesis, we use the term dimension instead, as the framework is not the core focus of this paper. One can think of dimensions as a space that layers belong to. This property is useful for studying the relationship between different layers in a multilayered network. The total number of layers in a multilayer network
is directly related to the number of dimensions and elementary layers in each dimension. For example, if a multilayer network has two dimensions with two elementary layers in each dimension, the total number of layers in the multilayer network equals four. This particular example is visualised in Figure 1 panel (b).

The set $V_M$ refers to the set of node-layer tuples (nodes) a subset of the Cartesian product of the set of vertices and the set of layers, i.e. $V_M \subseteq V \times L$. Simply put, a node-layer tuple describes which layer(s) a node exists in the multilayer network. Note that nodes ($V_M$) and vertices ($V$) are distinct entities, every element in $V$ has to be present in at least one of the layers which is not true for every element in $V_M$. Every layer has its own set of nodes, and the union of those sets is $V$. Finally, $E_M$ is the set of edges, a subset of all possible combinations of node-layer tuple pairs, i.e. $E_M \subseteq V_M \times V_M$. However, there are two different types of edges in a multilayer network, intra-layer edges and inter-layer edges. Intra-layer edges connect nodes that belong to the same layer, and inter-layer edges connect to different layers. These two types of edges are powerful when analysing the relationship between the nodes in a multilayer network. One can think of intra-layer edges as the edges within a multilayer network that form a traditional graph inside a layer. In contrast, the inter-layer edges add new properties to the network structure. In order to understand the application of the beforementioned definition of a multilayer network, this section includes an explanation of two general multilayer networks with different number of dimensions.

![Figure 1](image_url)

**Figure 1:** Two distinct multilayer networks to explain the distinctions in structure and some of the terminology from the framework. One has nodes and edges, while the other does not. Panel (a) shows a multilayer network with one dimension and three layers ($\alpha$, $\beta$ and $\gamma$), whereas panel (b) displays a multilayer network with two dimensions and four layers and was inspired by Kivelä et al. [4].

In the multilayer network shown in Figure 1 panel (a), there are a total of five
2 BACKGROUND AND PRIOR WORK

vertices. This set of vertices is denoted as \( V = \{1, 2, 3, 4, 5\} \) in the quadruple \( M = (V_M, E_M, V, L) \). The network has only one dimension, which can be visualised as a y-axis to which the layers belong to. The elements of the set \( L \) is simply listed as \( L = \{\alpha, \beta, \gamma\} = L_1 \) in the quadruple. Since \( d = 1 \) the set of layers is simply \( L_1 \) according to the definition of elementary layers in the framework. Inter-layer edges are shown as dotted lines, whereas intra-layer edges are shown as solid lines. The combination of these edges makes up the set \( E_M \). The network comprises nine nodes, and the set \( V_M \) (node-layer tuples) indicates their locations. The node-layer tuples are:

\[
V_M = \{(1, \alpha), (1, \beta), (1, \gamma), (2, \alpha), (3, \alpha), (3, \beta), (4, \beta), (5, \beta), (5, \gamma)\} \subseteq V \times L
\]

For instance, the subset \( \{(1, \alpha), (1, \beta), (1, \gamma)\} \) implies that vertex 1 exists as three separate nodes in all three layers, \( \alpha, \beta, \) and \( \gamma \).

The multilayer network shown in Figure 1 panel (b) has a total of four layers and two dimensions \( (d = 2) \). However, there are no nodes or edges present in this example to demonstrate the structure of a multilayer network with more than one dimension. The two dimensions can intuitively be seen as a x-axis and a y-axis that consist of two elementary layers each. In this case, the elementary layers \( A \) and \( B \) make up the y-axis and the elementary layers \( C, D \) make up the x-axis. Using this way of thinking one can come up to a conclusion that the two sets of elementary layers for the dimensions are \( L_1 = \{A, B\} \) and \( L_2 = \{C, D\} \). Calculating the Cartesian product one can compute the set of layers to be \( L = \{(A, C), (A, D), (B, C), (B, D)\} \).

2.3 Temporal networks

Temporal networks are defined as networks that change their connections between their nodes over time [8]. One can think of temporal networks as a sequence of monoplex networks that illustrates how the overall structure of the network evolves by analysing a set of snapshots.

Figure 2: A temporal network \( T \) represented as a sequence of undirected graphs where the connections change over time. Every graph in the sequence represents a time instance as the time is scaled from left to right along the x-axis.
Figure 2 illustrates how the set of connections (set of edges $E$ for each graph) changes when analysing the three different time instances $t_0, t_1$ and $t_2$ in the temporal network $\mathcal{T}$. The set of vertices are labelled numerically $V = \{1, 2, 3, 4, 5\}$ and all vertices exist in all time instances. A pattern can be found by analysing the set of nodes and edges in each graph for the respective time instance. For example, the entity represented by vertex 3 has been connected to the most number of other entities as the graph evolves to its final state.

Temporal networks have been applied in a wide variety of fields, such as modelling the spread of diseases and transportation systems [9, 10]. For example, in 2021, a meta-population model was published, and it was based on temporal networks for modelling the spread of the COVID-19 virus in Italy [11]. The model was adjusted using epidemic data from the first wave of the COVID-19 outbreak in Italy. The model aimed to predict how countermeasures such as social distancing would affect the spread of the virus. This study used temporal networks to analyse the evolution of official virus deaths as restrictions were implemented, in order to predict future behaviours.

![Layer 1](image1.png) ![Layer 2](image2.png) ![Layer 3](image3.png)

**Figure 3:** The temporal network in Figure 2 as a multilayer network. Every time instance is mapped to a layer resulting in a multilayer network with three layers.

Temporal networks can also be represented as multilayer networks, where each layer represents a time instance $t \in \mathcal{T}$ [4]. If there is only one dimension (i.e. $d = 1$), then the dimension in question is time, where each layer is an instance $t$ according to a time scale. Figure 3 is a simple example of how a multilayer network can be constructed from the three time-instances in Figure 2. However, temporal multilayer networks are not restricted to one dimension (time); additional dimensions can be added to represent additional types of edges in the network. Temporal multilayer networks are likely multiplex too, to analyse the layers’ connections over time. The general framework provided by Kivelä et al. can describe multilayer networks that are both temporal and multiplex. Temporal networks are relevant when discussing the dataset chosen for constructing
multilayer networks in this thesis. This point will be elaborated on in Section 3.2.

2.4 Multiplex networks

Multiplex networks have multiple types of edges to show different connections between the nodes. The reason for constructing networks with multiple types of edges is due to the fact that using only one type of edge restricts the possibility of information discovery and the understanding of the complex systems being modelled. Multiplex networks can be modelled using multilayer networks through the use of layers. The layers will usually contain the same node sets, where each layer focuses on representing a type of interaction between repeated nodes in each layer [4]. The inter-layer edges can be thought of as a method of “keeping track” of entities across different layers. In contrast, the intra-layer edges represent a different type of connection.

Figure 4: A multiplex network $\mathcal{M}$ represented as a multilayer network with three layers ($A$, $B$ and $C$).

The network $\mathcal{M}$ in Figure 4 is a multiplex network that consists of the same set of vertices $V$ as the general example in Figure 1 panel (a). As before, the intra-layer edges are marked by solid lines and the inter-layer edges are marked by dotted lines. It is important to note that all nodes are present in all layers; this means that $\mathcal{M}$ is node-aligned [4]. However, some of the software used in this thesis requires multiplex networks to be strictly node-aligned which is not always the case. This problem can be fixed by adding the “missing” nodes to the layers until the multiplex network is node-aligned. For more details on
which software requires node-aligned multiplex networks, see Section 4.

One can see that the inter-layer edges in $E_M$ connect the same entities in both layers. Inter and intra-layer edges usually illustrate two or more different types of connections, depending on what the multiplex network is modelling as a whole. Going back to the general framework by Kivelä et al. [4], multiplex networks are defined as a sequence of graphs:

$$\{G_i\}_{i=1}^b = \{(V_i, E_i)\}_{i=1}^b$$

where $i$ is used as an index for the sequence of graphs and $E_i \subseteq V_i \times V_i$ is the set of edges. In this example, the multiplex network $M$ is a sequence of three graphs $G_A = (V_A, E_A), G_B = (V_B, E_B), G_C = (V_C, E_C)$. Additionally, note that in this example $V_A = V_B = V_C$ which does not have to be the case in a multiplex network.

### 2.5 Supra-adjacency matrices

An adjacency matrix describes which vertices are adjacent in a graph $G$. However, multilayer networks can be described similarly, using supra-adjacency matrices [4]. Supra-adjacency matrices consist of all adjacency matrices of the graphs in a multilayer network $M = (V_M, E_M, V, L)$. In other words, all of the layers in $M$ are considered in the supra-adjacency matrix, yielding a matrix where all nodes and edges are included.

![Figure 5](image.png)

**Figure 5:** A multilayer network with two layers (right) and its supra-adjacency matrix (left). The visualisation of this example was created using multiNetX [6]. All of the nodes in the multilayer network are labelled as consecutive integers and the two layers are distinguished by the inter-layer edges. The supra-adjacency matrix distinguishes between intra-layer and inter-layer edges in the multilayer network by using black and grey-coloured fields, respectively.

In Figure 5’s multilayer network, if we assume that the inter-layer edges connect the same entities, in other words, the tuples $(0, 5), (1, 6), \ldots, (4, 9)$ represent five vertices, then those edges are referred to as coupling edges. These couplings
are also considered diagonal when all of the inter-layer edges connect nodes and their corresponding counterparts in other layers [4], which is also true if the above assumption is made. If these assumptions hold, then the multilayer network can be classified as multiplex. This classification can be made since diagonal couplings are a characteristic of multiplex networks according to the framework used. In addition, the inter-layer connections in the supra-adjacency matrix are arranged diagonally due to the couplings being diagonal. This is evident in the corresponding supra-adjacency matrix for the multilayer network shown in Figure 5.

2.6 Aggregation

When dealing with multilayer networks, network aggregation techniques can be employed to compress the network and obtain a new version of it [12]. One particular technique is important to cover for this thesis and that is “flattening”, which will just be referred to as aggregation to stay consistent with the terminology used in the multilayer network software. When aggregating a multilayer network, the layers are combined using a process similar to the OR logical operator. This results in a monoplex network where the original set of vertices in the multilayer network are maintained (nodes are combined into a single vertex). The edges in the monoplex network are the union of all edges connecting the nodes in \( M \). The example in Figure 6 shows how an aggregation technique is used on one of the software being analysed in this thesis.

There are various types of aggregation operations, including weighted, unweighted, and layer aggregation. The selection of the technique depends on how much information is intended to be retained from the multilayer network in the monoplex one generated. More information about these aggregation based techniques mentioned can be found in [12].

---

**Figure 6:** A node-aligned multilayer network with two layers A (left) and B (middle) aggregated to a monoplex network (right). The visualisation of this example was created using uunet.multinet [5]. Each layer is represented as a cell within a grid. The monoplex network is formed from the aggregation of all nodes and intra-layer edges from the input multilayer network. This can be thought of as an OR logical operation on the layers.
2.7 Degree measurements and neighbourhood

In a monoplex graph $G = (V, E)$, the degree of a vertex $v \in V$ refers to the number of edges that are connected to it [13]. The neighbourhood of a vertex $v$ is defined as the set of vertices $N \subset V$ that can be reached from $v$ [14]. The set $N$, therefore, consists of the adjacent vertices to $v$. The degree of the vertex $v$ is the same as the number of adjacent vertices or neighbours ($|N|$) unless there is a self-looping edge. A self-loop is an edge that connects a vertex to itself in the graph $G$. If a vertex has a self-loop, it is counted twice and increases the degree of that vertex by two. In the context of a directed graph, it is common to assess the in-degree and out-degree of each vertex, which correspond to the number of incoming and outgoing edges, respectively. These metrics prove valuable in several applications, such as gauging influence (where a higher in-degree indicates greater influence), identifying directed relationships, and tracking information flow.

Degree measurements become particularly useful after aggregating a multilayer network, resulting in a monoplex network. In this scenario, the degree of a vertex in the aggregated network corresponds to the degree of that vertex in the multilayer network.

2.8 Layer difference

Given a temporal multilayer network with one dimension, finding the unique set of edges from two or more layers of the network can be useful in order to identify anomalies and monitoring how the sequence of events changes over time in terms of unique connections being removed or added. This operation is based on neighbours-XOR which is a computational method for finding the set of neighbours for a vertex in a given set of layers that are not present in the other layers of the multilayer network [4]. This operation is utilised in order to compute a generalised exclusive or operation $M_{\oplus}$ (XOR) to find the total set of unique outgoing edges for every vertex in the multilayer network. The resulting monoplex network $G_{\oplus}$ constructed using the resulting unique set of edges is denoted as the layer difference.

![Figure 7](image)

Figure 7: The layer difference as a monoplex network of a temporal multilayer network with two layers where the intra-layer edges are unweighted and directed. The removed and added edges are displayed separately when transitioning from layer A to layer B. Finally, the resulting layer difference monoplex network is shown.
3 Methodology

Four different multilayer network software were compared using a multilayer network model. The model contains citation network data using a publicly available dataset from a past article; therefore, this thesis will not discuss any data scraping or collection methods. This dataset was chosen for this comparative analysis to evaluate the software’s performance, scalability, and durability under real-life scenarios.

3.1 Dataset overview

The chosen dataset contains citation data on journals over time [15]. The data spans between 1900-2013, and all citation data has been divided into time windows portraying which journals are active during that time window and their citation patterns. A journal is considered active in a time window if it has publications in the given time period. Every time window contains data on which journal has been cited by whom. A citation in this context means that an article in a journal has cited another article belonging to either another journal or the same one. Journals will also be referred to as vertices when modelling parts of the dataset as a multilayer network. However, nodes will not refer to the same characteristic as vertices. There is one vertex for every journal in the multilayer network, but several nodes in different layers will usually represent that journal in different time windows. See Section 3.2 for more details.

The dataset is divided into two different time windows; for the years 1900-1970s, the time window is ten years meaning that each time window in this range is an aggregated network of citations for a ten-year period. After the 1970s, the data is divided into 5-year time windows meaning that the aggregated networks constructed from one of these subsets of the dataset illustrate citations between active journals within that five-year time window. The data has been collected within 5-year time windows after the 1970s due to the number of journals and citations growing exponentially.

The relationship between the active journals over time and their corresponding citation patterns was modelled using a temporal multilayer network. This multilayer network model is elaborated on and visualised in Section 3.2. The active journals are the network nodes, and the intra-layer edges describe the citation pattern between the active journals for the respective time instance. The time instances were analysed chronologically, and the time window changed depending on what subset of the dataset was being analysed (before or after the 1970s).

3.1.1 Data format

From each time window in the dataset, there are two mandatory files and an optional one to construct an aggregated directed or undirected network. The first file contains the set of vertices where the line number corresponds to its node ID, and the line’s content is the journal’s name, which can also be used
3 METHODOLOGY

to identify the node or be used as a node label. Next, a separate file utilises the defined node IDs to represent the network’s edges in the following format for each line: sourceID targetID weight. It should be noted that whitespace separates each edge property, creating three different fields in the file. Directed edges can be created from this format, as each line represents an outgoing edge from a source and target node.

Additionally, weights can be added to the edges where appropriate using the third field of each line. The weight in the context of these citation networks is the number of times an article belonging to a journal has cited an article from another; therefore, weight > 0. It is also worth noting that the list of edges for a time window can also consist of self-loops, stating that an article has cited another article in the same journal. To summarise the format of the nodes and edges in a time window, every line is an edge stating that an article in journal sourceID cited an article in journal targetID, and there are a total of weight of such citations between them.

<table>
<thead>
<tr>
<th>journals.txt</th>
<th>citations.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td># node</td>
<td># stw</td>
</tr>
<tr>
<td>A</td>
<td>0 1 1</td>
</tr>
<tr>
<td>B</td>
<td>0 2 2</td>
</tr>
<tr>
<td>C</td>
<td>1 2 6</td>
</tr>
<tr>
<td>D</td>
<td>1 3 2</td>
</tr>
<tr>
<td></td>
<td>2 3 2</td>
</tr>
</tbody>
</table>

Figure 8: A directed weighted graph $G = (V, E)$ created using two text files defining its structure. The file journals.txt contains four lines specifying the network nodes; $V = \{A, B, C, D\}$, and the second file citations.txt contains five lines specifying its set of directed weighted edges $E$ using the line number of the other file as node IDs. The two example files follow the same format as the data in the dataset defines the aggregated citation networks for every time window, where the line numbers are used to identify the active journals.

The format of the nodes and directed weighted edges in the files defined in Figure 8 is a simple example of aggregated citation network creation from a time window in the dataset. In this example, the nodes are labelled alphabetically instead of using journal titles for simplicity to comprehend and visualise the construction of the networks. For further analysis, each time window contains a file categorising every journal into a particular field as an attribute for each journal. For example, a subset of the journals within a time window is assigned to Biology, whereas another subset is assigned to Physics. This information comes from an analysis using stochastic block models from the original article that provided this selection of citation networks [15]. However, this thesis does not explore stochastic block models nor details how the journals were categorised.
into these fields. Instead the dataset containing citation network data was utilised as a resource for comparing multilayer network software and discovering additional information that was not directly provided by the dataset.

On the other hand, importing the beforementioned data from the relevant time windows in the dataset and creating a multilayer network using the libraries is time-consuming. Especially if repeatedly done every time executing a program that manipulates or visualises a large multilayer network. Therefore, the constructed multilayer networks were created once using the raw dataset and then saved to a custom data format which is explained more in detail in Section 3.2.1. Some advantages of reformatting the raw data are that new useful specifications can be declared in a compact manner. For example, node-layer tuples, node labels, additional node attributes, intra and inter-layer edges.

3.2 Multilayer network model

A temporal multilayer network model was chosen to analyse how active journals and their citation patterns evolve using the given dataset. Each layer in the network corresponds to a directed graph representing the active journals and their citation patterns (intra-layer edges) for a time window. The inter-layer edges connect the active journals in the next time instance, indicating that those specific journals are active in both time windows. Therefore, if a journal $J$ has been active in all time windows, there will be an inter-layer edge between every pair of layers that are vertically “stacked” in the multilayer network connecting the nodes that correspond to $J$.

![Figure 9](image.png)

**Figure 9:** Overview of the temporal multilayer network’s layer and dimension specifications for modelling the chosen dataset. Each layer corresponds to an aggregated citation network containing active journals within the time window as the nodes and their citation patterns as intra-layer edges.
3 METHODOLOGY

Figure 9 displays a multilayer network model where every layer represents a time window of either five or ten years. One dimension is present in the multilayer network model corresponding to time \((d = 1)\), and the order of the layers allows the citation networks to be analysed chronologically (in this case, from top to bottom). In order to test the scalability of certain operations using multilayer network software, layers are added incrementally from the 1900s to 2005-2010, with no overlap in years within the layers. It is important to note that while the last available time window (2008-2013) was not included in scalability testing, it was still accessible from the model.

3.2.1 Converting the dataset to multilayer network format

```
# Number of Dimensions: d
1
# Vertex info: <vertex>,<layer>
0,1910
0,1920
0,1930
1,1920
1,1930
2,1930
3,1920
3,1930
4,1910
...

# Intralayer Edge Info: <source>,<layer> -- <target>,<layer>
0,1910 -- 89,1910
0,1910 -- 35,1910
0,1910 -- 95,1910
0,1910 -- 70,1910
0,1910 -- 157,1910
0,1910 -- 260,1910
0,1910 -- 235,1910
0,1910 -- 274,1910
...
```

Figure 10: An example of a file describing the specifications and data of a multilayer network. This custom data format illustrates how multilayer networks were imported and exported in Python. Lines starting with a hash symbol mark three main components of the data: node-layer tuples, intra-layer and inter-layer edges. Vertex Info: All node-layer tuples of the multilayer network. Intralayer Edge Info: All intra-layer edges connect a source and target node in the same layer. Interlayer Edge Info: Inter-layer edges connect nodes in different layers meaning that the layers of a source and target node cannot be the same.

Figure 10 is an example of how multilayer networks are defined and outlined from the dataset using a custom data format. It merges several citation networks into a multilayer network by assigning each time period to a layer and
adding inter-layer edges to link the active journals across different layers. This format was designed to make the process of importing multilayer networks into libraries more straightforward by having all types of edges and node-layer tuples already defined. However, while consecutive integers were used in the example for simplicity, they were not used in the implementation. The aim was to retain the journal title for every node. One potential approach to achieve this is by assigning journal titles as node IDs. However, this technique could consume more memory while creating multilayer networks. The node IDs would be in string format instead of integers, depending on the underlying implementation of the sequence of graphs for the layers. To address this issue, a global ID was used instead in order to lookup the journal title for every node and vertex when needed. The following shell command was first executed to gather all unique journal titles from the dataset, which were then sorted alphabetically and saved in a separate file:

```
find /PATH_TO_DATASET_DIR/ -name "blocks_level_0.txt" -type f
--exec cat {} \; | sort | uniq > output.txt
```

To run the shell command, first the original dataset has to be downloaded. Furthermore, the placeholder path, `/PATH_TO_DATASET_DIR/`, needs to be replaced with the actual path to the dataset’s parent directory. The command searches for files named `blocks_level_0.txt` within the dataset, as each time window has a stochastic block-level grouping of journals into subjects. Block level 0 contains the journal titles without any grouping, which was all that was needed for this thesis. Appendix A contains a condensed version of the dataset directory output that was generated by running the `tree` command. In addition, there is a table that provides information on the contents of each file within each time window and how they relate to the creation of multilayer networks based on the pre-established model.

<table>
<thead>
<tr>
<th>ID</th>
<th>Journal Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Journal Title 0&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Journal Title 1&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Journal Title 2&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;Journal Title 3&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;Journal Title 4&quot;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>62487</td>
<td>&quot;Journal Title 62487&quot;</td>
</tr>
<tr>
<td>62488</td>
<td>&quot;Journal Title 62488&quot;</td>
</tr>
<tr>
<td>62489</td>
<td>&quot;Journal Title 62489&quot;</td>
</tr>
<tr>
<td>62490</td>
<td>&quot;Journal Title 62490&quot;</td>
</tr>
<tr>
<td>62491</td>
<td>&quot;Journal Title 62491&quot;</td>
</tr>
</tbody>
</table>

**Figure 11:** An example of how a node’s ID in a multilayer network is transformed into its corresponding journal title by utilising a constructed data frame.
After gathering and sorting all the distinct journals, their line numbers were assigned as global IDs. These IDs were then used by all the nodes and vertices in the multilayer network. To facilitate this process, a data frame data structure was utilised in Python and R to convert the IDs to journal titles and vice versa. Figure 11 illustrates the conversion of a node ID from a multilayer network to its corresponding journal title. For simplicity, the journal titles are numbered the same as the node IDs in Figure 11. Therefore, the title of the journal assigned to node 4 is “Journal Title 4”.

![Multilayer Network](image)

**Figure 12:** A multilayer network constructed and visualised using Pymnet [4], modelling a subset of the chosen dataset.

The multilayer network in Figure 12 is an example of how the dataset will be modelled throughout the thesis and used to visually grasp the size of the dataset being modelled. The black nodes illustrate active journals, and the three layers indicate the evolution of active journals in the 1910s, 1920s and 1930s. Intra-layer edges are the citations and are represented by solid grey lines. Inter-layer edges are represented by dotted grey lines illustrating that an active journal is also present in another layer/time-period. Visualising temporal multilayer networks from the model can be difficult because of its numerous layers. Thus, it is essential to simplify the networks or concentrate on a subset of the layers to emphasise specific characteristics of the networks. Furthermore, if any journals seem to have no connection to any other journals when plotting a subset of the dataset (also known as isolated nodes) it means that they are connected through a self-looping edge. As a result, they are still considered active and included in their respective layer. However, for the sake of simplicity, self-looping edges are not displayed in any plots.

---

[A data frame is a two-dimensional data structure similar to a table in a database [16].]
4 Multilayer network software

The four multilayer network software that were compared in this thesis are Pymnet, uunet.multinet (Multinet), multiNetX and MuxViz [4, 5, 6, 3]. These four libraries were selected because they are commonly used by professionals for analysing and visualising multilayer networks. However, the overall features and techniques for information discovery differ for this selection of software; this section summarises what each software provides and its purpose. On another note, most of the above software is still under active development, or has not been updated in years. Therefore, the following overview is fully based on the most recent documentation and source code available as of writing this thesis. A comparison of relevant key features available between Pymnet, Multinet, multiNetX and MuxViz can be found in Table 1. Further information about the software is summarised in their corresponding sections.

Table 1: Comparison of key features provided by the selected multilayer network software. Network Visualisation: A constructed multilayer network can be plotted. Directed Networks: There is support for using directed graphs as the layers of a multilayer network. Further calculations within the library and visualisation will adapt to this property. Simple Diagnostics: Possibility to calculate a summary of the overall size and specifications of the multilayer network in terms of the number of vertices, intra-layer edges, inter-layer edges, nodes and layers. Community Detection: Software provides the ability to execute community detection algorithms on the network. Degree measurements: Degree distribution can be calculated for the individual layers or for all $V$. Network Density: The network density can be calculated for the multilayer network, which measures how connected a network is compared to the network’s fully-connected potential. Aggregation: Software provides a method for aggregating a multilayer network into a monoplex network. Supra-adjacency Matrix Representation: A multilayer network can be created, visualised or converted into a supra-adjacency matrix. Read Network From File: Multilayer networks can be created from a predefined specification in the form of a file.

<table>
<thead>
<tr>
<th>Features</th>
<th>Pymnet</th>
<th>Multinet</th>
<th>multiNetX</th>
<th>muxViz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Visualisation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Directed Networks</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Simple Diagnostics</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Community Detection</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Degree Measurements</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Network Density</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Aggregation</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Supra-adjacency Matrix Repr.</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Read Network from File</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
4.1 Pymnet

Pymnet has been implemented using solely Python code and is based on the chosen framework for analysing general multilayer networks. Additionally, it has been mainly developed by one of the authors of the article, who defined the multilayer network framework used in this thesis.

The library provides two new data structures; one for general multilayer networks based on the built-in Python dictionary data structure\(^2\), and the other for multiplex networks as a sequence of graphs. Visualising multilayer networks in Pymnet is handled by an additional library called Matplotlib, which is used for creating and plotting interactive figures and saving them as files in various formats \([17]\). Finally, when working with monoplex networks, the library can utilise NetworkX operations. NetworkX is a popular Python package for analysing complex networks \([18]\). Functions provided by NetworkX are useful when transforming Pymnet multilayer networks to monoplex networks for further analysis. Pymnet provides functions for transforming multilayer networks. For example, converting a multilayer network into a supra-adjacency matrix, creating a sub-network given a set of layers and nodes, and reducing the number of dimensions of the multilayer network through aggregation. Additionally, network analysis methods such as degree distributions and network density can be calculated using the library.

According to the library’s documentation, the reasoning for implementing two fairly similar data structures is that the inter-layer edges are not the focal point of multiplex networks and should not explicitly be stored inside the multiplex data structure. Rather than storing the inter-layer edges, they are dynamically generated when needed. For example, when visualising a multiplex network, the inter-layer edges are generated using a set of rules. This design decision can reduce memory consumption when constructing a multilayer network that is multiplex (depending on the number of inter-layer edges in the network).

The data structure for general multilayer networks is made up of a Python; dictionary of dictionaries. The outer dictionary stores all the nodes in the multilayer network as keys; its adjacent nodes are values for this external dictionary. In comparison, the inner dictionary consists of those adjacent nodes as keys and the weights connecting them as values. This particular data structure for a multilayer network ensures that accessing, inserting, and deleting nodes and edges can be done with an average time complexity of $\Theta(1)$. The maximum amount of space needed for $M$ is determined by its worst-case space complexity, which is $O(|V_M| + |L| + |E_M|)$. This is usually dominated by the number of edges in $M$, as both intra and inter-layer edges are stored in the general multilayer network data structure \([4]\).

\(^2\)The dictionary data structure in Python is in the form of a hash map consisting of key-value pairs.
4.2 Multinet

The Multinet library used in this analysis is a Python porting from the original Multinet R library, first released in 2017 [5]. Additionally, Multinet is also available in the programming language C++. However, in this thesis, the Python porting of Multinet was experimented with to keep the implementation as consistent as possible. Multinet was designed for social network analysis and provides various features for analysing multilayer networks similar to Pynmet. Multinet can construct multilayer networks in two ways; reading the network from a file using a format supported by Multinet and first creating NetworkX graphs and using those graphs as the layers to create a complete multilayer network. Multilayer networks can also be aggregated into a monoplex network and exported as NetworkX graphs as Pymnet. Multinet also utilises matplotlib for plotting multilayer networks layer-by-layer in a grid. Multinet also provides several methods of analysing the structure of multilayer networks. For example, degree distribution, layer comparison, layer summary and extracting distinct adjacent nodes from a vertex. Additionally, Multinet provides four different community detection algorithms for finding sets of closely connected vertices in the multilayer network. Further information about community detection algorithms can be found in [19].

Multinet stands out from other software, employing a database design approach for storing edges and vertices in a multilayer network. Two primary data structures, vertex and edge cubes, collectively called multilayer cubes, are utilised to construct a multilayer network.

4.3 multiNetX

multiNetX is a python package based on the beforementioned NetworkX library for creating, visualising and manipulating multilayer networks [6]. A multilayer network can be built by providing a supra-adjacency matrix or a list of NetworkX graphs as the layers. The supra-adjacency matrix is mandatory to define the inter-layer edges of the multilayer network. Due to this package inheriting most functionality from an existing library, fewer features are available than Pymnet and Multinet. For example, there are no built-in operations for multilayer network analysis except the overall specifications of the multilayer network. For instance, multiNetX can calculate the number of intra-layer edges, inter-layer edges, vertices and nodes. However, to conduct further analysis, it is necessary to first convert the layers of the multilayer network into NetworkX graphs and do additional operations on those graphs. multiNetX also utilises Matplotlib for visualising multilayer networks but also supra-adjacency matrices. The data structure for multilayer networks provided by multiNetX consists of a list of NetworkX graphs that can be either directed or undirected, as well as a inter-adjacency matrix.\(^3\)

---

\(^3\)The inter-adjacency matrix defines the inter-layer edges of the multilayer network and is used to construct a complete supra-adjacency matrix and plotting multilayer networks with inter-layer edges.
The interconnections of the multilayer network are defined using a List of Lists (lil) sparse matrix. A sparse matrix is a matrix which consists mainly of elements that are zero [20]. A lil sparse matrix is a compressed format of a sparse matrix using lists of lists for looking up values. Every row in the lil sparse matrix is a list containing column indices and non-zero values. Usually the lists are sorted by column index for faster lookup in the matrix [21]. The advantage of using lil sparse matrices is that the total amount of memory required is not dependent on the size of the matrix, but the number of non-zero elements by storing the column index and non-zero values for each row.

Due to multiNetX utilising NetworkX objects for layer construction, it is also crucial to understand the underlying data structure for general NetworkX graph objects. The underlying data structure is based on the adjacency list representation of a graph and implemented using a Python dictionary of dictionaries data structure [18]. The outer dictionary stores the nodes as the keys and its adjacent nodes as values in the form of a dictionary as well. The values of the inner dictionary contains the attributes of that edge, such as the weight if it is a weighted graph. NetworkX provides different classes for graphs depending on the properties desired, for example, there is a separate class for directed graphs where the order of the edge pair matters compared to an undirected graph. All graph operations are defined as methods for each class.

4.4 muxViz

muxViz is an R package for analysing and visualising multilayer networks with a primary focus on multiplex networks [3]. Unlike the previously mentioned software, muxViz also provides a graphical user interface (GUI) for constructing and analysing multilayer networks. The GUI is a convenient tool for users who want to analyse multilayer networks already in a specific data format supported by muxViz for importing multilayer networks.

In this analysis, multilayer networks were created in muxViz by writing the ones already constructed in Python to the custom muxViz data format. When creating a multilayer network in muxViz using files in muxViz data format, a list of valuable properties about the multilayer network is returned. For example, a list of graphs representing the layers, a supra-adjacency matrix and other node and layer attributes such as labels and IDs are stored separately in this list.

Alternatively, a multilayer network can be imported into muxViz using NetworkX in Python. This can be achieved by exporting the layers as a list of graphs from files in GraphML (Graph Markup Language) format. GraphML is a file format based on XML (Extensible Markup Language) for sharing graph structures and their data [22]. The following steps describe the process of creating the layers of a muxViz multilayer network using GraphML files:

1. Create a list of NetworkX graph objects in Python along with their respective layer names defining the layers of the multilayer network without inter-layer edges.
2. For every layer, write the graph to a file in GraphML format using the `write_graphml()` method provided by NetworkX.

3. In R, read the GraphML files and name of the layers, read the desired layers as `.graphml` files using the R package `igraph` and store the resulting graphs as a list.

4. The list of `igraph` objects can now be used as a base to create a multilayer network in muxViz.
5 Comparative analysis

The different features of Pymnet, Multinet, multiNetX, and muxViz were tested to analyse multilayer networks, and additional information from the dataset was extracted. Furthermore, accuracy and consistency were ensured by comparing the results of all software in the same scenarios. This helped in understanding the process of analysing multilayer networks and the potential differences in approach between different software.

Additionally, in order to evaluate the performance of the software more in detail, two separate sets of tools were utilised since the libraries are written in different programming languages - Python and R. These tools were chosen based on two distinct methods of comparing the libraries in more detail. The first method involves measuring the execution times of specific multilayer network operations as the load (multilayer network size) increases to test the scalability, while the second method involves profiling the programs implemented that use the respective libraries. Profiling the programs allowed for an analysis of the libraries’ overall behaviour, including their use of underlying multilayer network data structures and memory consumption.

5.1 Test environment

A MacBook Pro M1 running macOS Ventura was used to carry out the comparison analysis. The machine had Python 3.9.16 and R 4.2.3 installed, along with the necessary package dependencies for the multilayer network software selected.

5.2 Tools used

Python includes a highly beneficial module known as timeit that is capable of measuring the execution time of Python functions or code. It even permits the execution of a function or code snippet multiple times, which proves helpful in calculating the average of a list of execution times for a specific operation. Furthermore, the option to disable garbage collection with timeit ensures that any overhead that might influence timing outcomes is eliminated. The performance of various multilayer network operation implementations written in Python (Pymnet, Multinet, and multiNetX) were compared using timeit with garbage collection disabled.

In order to accurately measure the execution time of multilayer network operations in muxViz, the R package microbenchmark was utilised [23]. By executing a function or code snippet multiple times and storing the resulting execution times in an R variable, the microbenchmark package allowed for the collection of relevant data for comparing muxViz with multilayer network libraries written in Python. This ensured precise measurement of execution time similar to timeit in Python.

To monitor the execution of the Python programs implemented, the cProfile module was utilised. This module provides the ability to track different types of
execution times. For example, the execution time for the entirety of a program and specific parts of it [24]. cProfile is also capable of generating additional statistics, such as the number of function calls per function, which came to good use for optimising the implementation. A tool called memory.profiler was used to monitor memory usage in Python [25]. This tool can track memory usage for the entire Python process or specific lines of code. The tool utilised for monitoring execution in R was the profvis package [26], which provides a comparable interface.

5.3 Queries and data exploration

Three different queries were designed and executed to compare the software in different use cases. In the simplified feature comparison table from Table 1, it was observed that the four software options had varying features. Therefore, custom implementations were written for those lacking certain necessary features to execute the queries. The purpose of writing custom implementations to extend the specific libraries was to determine the difficulty level in obtaining the same results as the software that provides those specific missing features.

5.3.1 Query 1: Longest active journals

The first query is related to the definition of node-layer tuples, defining the presence of a vertex among different layers as different nodes. The goal is to determine how many and which time window(s) all of the journals were active in, given a temporal multilayer network from the model. The list of journals obtained from the query contains three essential details for each element: the journal’s title, the number of time windows it was active in the temporal network, and the specific time windows it was active in. For example the 1930s and 1940s. Additionally, the query summarises how many journals were active in all time windows of the modelled period.

![Figure 13: A simplified example of the results expected after computing the “longest active journals” query on a temporal multilayer network from the model. Note that this is just an abstract example and not based on actual data from the dataset. For simplicity, inter-layer edges are not visualised, and the node IDs are consecutive integers, along with their corresponding journal titles.](image-url)
5 COMPARATIVE ANALYSIS

It is important to note that if this query is executed on a multilayer network that is node-aligned, the query result will be redundant, since all journals will be present in all layers. The temporal multilayer network in the general example shown in Figure 13 is not node-aligned.

Executing this query on a temporal multilayer network $M=(V_M, E_M, V_L)$ requires a set of specific operations to be carried out in a particular order to ensure success. The chosen implementation to perform this query is summarised below as a series of steps:

**Query 1** *Compute the longest active journals between a range of decades*

1. Import the multilayer network $M$ that corresponds to the time period of interest
2. Get all elements in $V_M$ (node-layer tuples) from $M$
3. Store all node-layer tuples in a data frame
4. Group the data frame of node-layer tuples by node ID, and replace every node ID with its corresponding journal title
5. Resulting data frame contains:
   (a) Column 1: Number of layers present (int)
   (b) Column 2: Layers present (ex. 1900s, 1910,...) (list)
   (c) Column 3: Journal title (string)
6. Sort the data frame by the number of layers present
7. Return the data frame by writing to file and print a summary of how many journals were present in all layers

5.3.2 **Query 2: Layer difference**

A monoplex network can be created by computing the layer difference of a multilayer network from the model, which shows the unique citations between journals for that time period. However, due to the large number of intra-layer edges in every layer, calculating the layer difference alone may not always be helpful. This query was implemented for the purpose of exploring a type of temporal network analysis method and can be extended further to find more valuable pattern insights.

To prioritise the distinct citation between various journals, the chosen implementation excludes self-loops meaning that not all of the journals active for the given time period are not necessarily present in the resulting layer difference monoplex network.

**Query 2** *Calculate the layer difference of the multilayer network between a range of decades and return the result as a monoplex network*

1. Import the multilayer network $M$ that corresponds to the time period of interest
5 COMPARATIVE ANALYSIS

2. Get $V$ from $M$ using operations provided by software
3. Create an empty monoplex network in the form of a directed graph $G$
4. for every $v \in V$ in $M$
   (a) Find the set of non-duplicate neighbours for $v$ (all layers considered)
   (b) Add the set of non-duplicate neighbours for $v$ as outgoing edges to $G$
5. Return $G$

5.3.3 Query 3: Influencing journals

The most influential journals in a specific time period are identified by analysing the citation patterns of all active journals. This process involves examining the in-degree of a journal, which is the number of other journals that have cited an article from it. By using this method, the journals with the highest in-degree can be determined. To guarantee accuracy, self-loops are excluded from the analysis since the focus is solely on identifying the journals with the most citations from other journals.

![Figure 14: A monoplex network as a directed graph and visualised using Gephi](image)

Figure 14: A monoplex network as a directed graph and visualised using Gephi [27]. The network illustrates the aggregated citation network for the 1900s. However, the journals cited the most (in-degree) in the time period are plotted as a larger node size and are assigned the warmest colour according to the maximum number of other journals that have cited a particular journal. Self-loops are omitted and arrows displaying the direction of the edges are also omitted in order to simplify the plot.

Figure 14 displays the process for determining the most influential journals for a particular time window from the dataset. This approach has been extended to analyse an entire temporal multilayer network to identify the most influential
journals over a period of time. To achieve this, the in-degree of each vertex was calculated across all layers, and the generalised steps are listed below.

**Query 3** Find the top \( n \) influencing journals between a range of decades

1. Import the multilayer network \( M \) that corresponds to the time period of interest
2. Aggregate \( M \) to yield a monoplex network \( G = (V, E) \)
3. Get \( V \) from \( G \)
4. Calculate the in-degree for all \( v \in V \) and store the result as a data frame
   (a) Column 1: Journal Title (converted using nodeID to Journal Title data frame)
   (b) Column 2: In-degree
5. Sort resulting data frame by in-degree
6. Return the sorted data frame result by printing first \( n \) elements and writing the whole sorted data frame to a file
6 Result

After successfully importing multilayer networks from the model to the different software, it became clear that not all queries could be correctly implemented for all four libraries due to their differences in provided features and design decisions. For example, finding influencing journals in multiNetX was not possible since the layers are strictly stored as undirected graphs. Similarly, multiNetX does not provide the possibility to aggregate a multilayer network. However, this problem was solved using an equivalent NetworkX operation as a replacement yielding the same result as the rest of the libraries. Once the queries were implemented where appropriate, two specific operations were used for scalability testing: calculating the degree\(^4\) for all nodes and multilayer network aggregation.

6.1 Query execution

After executing the three designed queries on various temporal networks using different software, it was important to analyse the accuracy of the results. This included verifying that the journals were active during the entire study period when executing the longest-active journals query. Additionally, confirming that the output of the top influential journals query was accurate compared to other sources that studied this very topic.

Query 1: Find the longest active journals between 1900-1970s.

| Am._J._Sci. | AMERICAN_JOURNAL_OF_ANATOMY |
| Am._J._Psychol. | Nature |
| Rev._Neurol. | Anthropos |
| J._Biol._Chem. | Biometrika |

Summary: 70 journals active in all time windows

Figure 15: Output of computing the longest active journals between the 1900-1970s using Pymnet, multiNetX and Multinet (|L| = 8). An unordered selection of twenty journals (out of 70) is displayed in the output inside of the text box.

\(^4\)In multiNetX, the degree calculation for all nodes in a multilayer network was completed successfully. However, the software only supports degree calculations, not in and out-degree calculations. As a result, the query for influencing journals could not be implemented for multiNetX.
To ensure the accuracy of the longest active journals query, research was conducted on several journals’ establishment date and activity duration. For instance, a few journals, including Journal of Biological Chemistry and Biometrika, were chosen from the query output to verify the correctness displayed in Figure 15. The Journal of Biological Chemistry and Biometrika have been active since their establishment in 1905 and 1901 respectively [28, 29]. As a result, they were correctly identified as active during the 1900-1970s time period in Figure 15. However, validating every journal from the output was not feasible for every query execution due to time constraints and the scope of the thesis. Therefore, it was assumed that the temporal network model and query implementation was functioning correctly.

The layer difference query was successfully implemented in all software. However, the unique citation patterns between the journals were not analysed any further from the obtained monoplex layer difference network. Appendix B shows an example of what a layer difference monoplex network looked like for the layers 1900s and 1910s.

Finally, the third query was implemented, identifying the top n influential journals in a temporal multilayer network. As mentioned earlier, this query could not be implemented in multiNetX unless solely NetworkX objects and operations were used instead, which would cause the multiNetX library to not be used at all.

Query 3: Find the top 20 influencing journals between 1900-2010.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Journal Title</th>
<th>In-Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Science</td>
<td>20855</td>
</tr>
<tr>
<td>2</td>
<td>Nature</td>
<td>20111</td>
</tr>
<tr>
<td>3</td>
<td>NoData</td>
<td>17717</td>
</tr>
<tr>
<td>5</td>
<td>Lancet</td>
<td>10024</td>
</tr>
<tr>
<td>6</td>
<td>R. Engl. J. Med.</td>
<td>9638</td>
</tr>
<tr>
<td>7</td>
<td>Phys. Rev. Lett.</td>
<td>9103</td>
</tr>
<tr>
<td>8</td>
<td>Ann. NY Acad. Sci.</td>
<td>8924</td>
</tr>
<tr>
<td>9</td>
<td>J. Biol. Chem.</td>
<td>8815</td>
</tr>
<tr>
<td>10</td>
<td>JAMA - J. Am. Med. Assoc.</td>
<td>8640</td>
</tr>
<tr>
<td>11</td>
<td>Br. Med. J.</td>
<td>7769</td>
</tr>
<tr>
<td>13</td>
<td>An. Chem. Soc.</td>
<td>7566</td>
</tr>
<tr>
<td>15</td>
<td>Appl. Phys. Lett.</td>
<td>7318</td>
</tr>
<tr>
<td>16</td>
<td>Anal. Chem.</td>
<td>6934</td>
</tr>
<tr>
<td>17</td>
<td>Phys. Rev. Lett.</td>
<td>9103</td>
</tr>
<tr>
<td>18</td>
<td>Sci. Am.</td>
<td>6848</td>
</tr>
<tr>
<td>19</td>
<td>Cancer Res.</td>
<td>6725</td>
</tr>
<tr>
<td>20</td>
<td>J. Clin. Invest.</td>
<td>6686</td>
</tr>
</tbody>
</table>

Figure 16: Output of computing the top 20 influencing journals between 1900-2010 using Pymnet, Multinet and muxViz (|L| = 14). The journal titles and their respective in-degree are displayed in the text box. The top 100 journals between 1900-2010 can be found in Appendix C.

According to a study from 2009, the top three influential journals up to year 2007 were found and written as follows: “Three journals have by far and away the most overall influence on science: Nature, PNAS, and Science, closely followed by the Journal of Biological Chemistry” [30]. When analysing the query output for influential journals illustrated in Figure 16, it becomes apparent that the journals occupying the top positions are predominantly identical with journals from the direct quote. On the other hand, the third-ranked journal from the
query, denoted as “NoData”, seems to deviate from this comparison and stands out as an outlier.

While not all journals were validated this way due to time constraints, only the top-ranked influential journals were analysed to guide the implementation process. At the end of the implementation process, the query produced the same output for Pymnet, Multinet, and muxViz based on their ranking and in-degree count. For example, there were some instances where the in-degree varied slightly. Initially, the self-loops were not removed which caused some inconsistency in the results. However, once self-looping edges were eliminated, the results became consistent.

6.2 Scalability testing

The focus was placed on two specific multilayer network operations to evaluate and compare the scalability of all four multilayer network software: degree calculation and aggregation. The load size for each operation in the software was increased gradually by adding more layers to the multilayer network. Furthermore, the execution time for each operation was measured for fifty trials for every load size. Only the function call in the code responsible for executing the operation was timed, the total execution time for the whole program was thus not measured during scalability testing.

Table 2: Time taken to aggregate various multilayer networks using Pymnet, Multinet, multiNetX and muxViz in seconds. The multilayer network specifications, such as the number of vertices (|V|) was calculated using functions provided by the multilayer network software. Every data entry for the time taken to aggregate a multilayer network is the mean of 50 trials using `timeit` and `microbenchmark`. Since the multiNetX library does not provide a method for aggregating a multilayer network as seen in Table 1, the equivalent `compose_all()` function was used instead from NetworkX to aggregate the layers of the network.

<table>
<thead>
<tr>
<th>Multilayer Network Specifications</th>
<th>Time Taken to Aggregate (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pymnet</td>
</tr>
<tr>
<td>1900-1910s</td>
<td>2</td>
</tr>
<tr>
<td>1900-1920s</td>
<td>3</td>
</tr>
<tr>
<td>1900-1930s</td>
<td>4</td>
</tr>
<tr>
<td>1900-1940s</td>
<td>5</td>
</tr>
<tr>
<td>1900-1950s</td>
<td>6</td>
</tr>
<tr>
<td>1900-1960s</td>
<td>7</td>
</tr>
<tr>
<td>1900-1970s</td>
<td>8</td>
</tr>
<tr>
<td>1900-1990</td>
<td>10</td>
</tr>
<tr>
<td>1900-2000</td>
<td>12</td>
</tr>
<tr>
<td>1900-2010</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 2 shows that the multilayer network’s size, measured by the number of nodes, edges, and vertices, increased exponentially after the 1970s, as expected. This growth impacted the time required to aggregate the multilayer networks, but the results varied drastically. The addition of the last two layers to the multilayer network caused Multinet to experience significant holds, taking over 600 seconds to aggregate on average. On the other hand, muxViz performed
well with this size and took less than 10 seconds to aggregate on average. After receiving these results, a comprehensive investigation was carried out on the source code of all libraries, with particular attention given to Multinet and muxViz.

**Table 3:** Time taken to calculate the degree of the nodes of various multilayer networks using Pymnet, Multinet, multiNetX and muxViz in milliseconds. The presence of “-” in the data entries means that the calculation of the degree for all nodes in that particular software and multilayer network was not possible.

<table>
<thead>
<tr>
<th>Multilayer Network Specifications</th>
<th>Time Taken to Calc. Degree (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pymnet</td>
</tr>
<tr>
<td>1900-1910s</td>
<td>2</td>
</tr>
<tr>
<td>1900-1920s</td>
<td>3</td>
</tr>
<tr>
<td>1900-1930s</td>
<td>4</td>
</tr>
<tr>
<td>1900-1940s</td>
<td>5</td>
</tr>
<tr>
<td>1900-1950s</td>
<td>6</td>
</tr>
<tr>
<td>1900-1960s</td>
<td>7</td>
</tr>
<tr>
<td>1900-1970s</td>
<td>8</td>
</tr>
<tr>
<td>1900-1990</td>
<td>10</td>
</tr>
<tr>
<td>1900-2000</td>
<td>12</td>
</tr>
<tr>
<td>1900-2010</td>
<td>14</td>
</tr>
</tbody>
</table>

A scalability test was performed for degree calculation, and the corresponding results can be seen in Table 3. The method employed for the aggregation test was also utilised in this case. On the other hand, a noticeable difference was that multiNetX and muxViz could not calculate the degree for all given multilayer networks. This was due to the processes being terminated by the operating system for multiNetX and an R error, causing the program to terminate prematurely for muxViz.
7 Discussion

After collecting all the data from the queries and scalability tests, there are still some unanswered questions that need clarification. The first question pertains to the highly ranked journal named “NoData” in the influential journals query result. Why is it there? Multinet performed poorly compared to other software during scalability tests. What caused this? Lastly, what were the reasons behind muxViz and multiNetX’s inability to compute the degree of some of the larger multilayer networks?

7.1 Anomaly investigation

In Figure 16, there was an anomaly found in the influencing journals query output that remains a mystery. However, a theory was proposed to shed some light on the presence of this journal title.

Upon analysing all time windows from the dataset, it was discovered that 26 files defining the titles of all active journals for various time windows (Blocks level 0) included a journal title labelled “NoData”. Therefore, the multilayer networks derived from the dataset exhibit the desired structure. At the time of writing the thesis, there was no information found about this anomaly in the dataset description or the study that provided the dataset. On the other hand, it is possible that an error occurred during previous processing, leading to a journal being mislabelled as “NoData”.

7.2 Potential overhead in Multinet

Upon analysing the source code of the Multinet library, it was discovered that the Python modules within it call upon certain C++ operations at specific points. The implementation of this was done through PYBIND11, a library that creates Python bindings for C++ code [31]. This type of implementation design was expected since the version of Multinet used is a Python adaptation of one of the original Multinet library implementations. According to the library’s documentation, structures might be copied between Python and C++11 whenever bound depending on how the library was utilised. This might be one of the reasons why Multinet struggles to perform as well as the other libraries when creating large multilayer networks since there is a possibility of additional overhead being introduced. Specifically, additional copying of large structures between Python and C++.

7.3 Exploring coercion issues in muxViz

Attempting to calculate the degree for all nodes in muxViz on one of the largest networks from the model resulted in the following error message:

```
In int2i(as.integer(i), n) :
  NAs introduced by coercion to integer range
```
7 DISCUSSION

Execution halted

This error message states that a function called from the muxViz library attempted to convert a value i to an integer causing the integer to be out of range. When an integer goes out of range in R, it is replaced with NA (not available). The unusual aspect of this error was that this error was not thrown when performing the aggregation scalability test for muxViz using the same multilayer networks. It is possible that the error occurred because the values from the degree calculation became too large due to an external factor. However, it is difficult to confirm this theory and it does not seem correct since previous results from the degree calculations in muxViz were correct when being compared to the results from all other libraries. By running the traceback() function in the R session, more details were given regarding where the execution halted.

```
> traceback()
16: int2l(as.integer(1), n)
15:  `<-(`*tmp`, i, value = 1)
14:  `<-(`*tmp`, i, value = 1)
13:  `<-(`*tmp`, Matrix::Which(A != 0), value = 1)
12:  `<-(`*tmp`, Matrix::Which(A != 0), value = 1)
11: binarizeMatrix(SupraAdjacencyMatrix)
10: FUN(X[[i]], ...) 
  9: lapply(i:Layers, function(x) SupraAdjacencyMatrix[(1 + (x - 1) * Nodes):(x * Nodes), (1 + (x - 1) * Nodes):(x * Nodes)])
  8: SupraAdjacencyToNodesTensor(binarizeMatrix(SupraAdjacencyMatrix), Layers, Nodes)
  7: GetMultiInDegree(SupraAdjacencyMatrix, Layers, Nodes, isDirected)
  6: GetMultiDegree(supra_matrix, num_layers, num_nodes, FALSE)
  5: microbenchmark(GetMultiDegree(supra_matrix, num_layers, num_nodes, FALSE), times = trials, unit = "ms") at muxViz_bm.R#55
  4: eval(ei, envir)
  3: eval(ei, envir)
  2: withVisible(eval(ei, envir))
  1: source("muxViz_bm.R")
```

The last function called from the muxViz library is the binarizeMatrix() function (step 11 in output). This particular function assigns every element in a sparse matrix into either 1 or 0. In this case, the sparse matrix is the supra-adjacency matrix. If an element is not zero, it is assigned to 1 (step 12). Else it is assumed that the element is already equal to 0. Remembering the error message thrown from the output, it seems like the supra-adjacency matrix contains elements that are NA values, causing the execution to halt inside the binarizeMatrix() function. It is still unknown why certain values are assigned NA when they exceed the integer range in the supra-adjacency matrix.

7.4 Case Study: multiNetX memory consumption

In contrast to the scalability test problems experienced in muxViz, the reasoning for why multiNetX could not calculate the degree for all nodes was due to exces-
sive memory consumption. The operating system clearly reported this, forcing the process to be terminated. To further analyse the situation, the last possible node calculation from the test was executed using multiNetX while tracking the memory consumption over time. Additionally, Pymnet and Multinet were also analysed under the same condition to compare their memory usage patterns.

---

**Figure 17:** Memory consumption in mebibytes over time measured using `memory_profiler` when calculating the degree for all nodes using multiNetX. This execution used the multilayer network representing the time windows 1900-1960s from the model ($|L| = 7$). The legend on the right side of the line plot shows the different functions executed. Each function is represented by a coloured square bracket, and they are laid out on the plot according to when they started and finished executing. An opening square bracket indicates the start of a function, while a closing square bracket indicates its completion. Peak memory usage: 4691.391 MiB

---

**Figure 18:** Memory consumption in mebibytes over time measured using `memory_profiler` when calculating the degree for all nodes using Multinet. Following Figure 17. Peak: 1270.188 MiB
Figure 19: Memory consumption in mebibytes over time measured using memory profiler when calculating the degree for all nodes using Pymnet. Following Figure 17. Peak: 611.953 MiB

After analysing the line plots created with memory profiler in Figures 17, 18, and 19, an unusual pattern was discovered in the multiNetX plot. When creating a multilayer network object in multiNetX, the memory usage spiked when calling the corresponding function and inserting the NetworkX objects into the data structure. Furthermore, the peak memory usage was significantly higher than that of other libraries operating under the same conditions. Recall that multiNetX requires a node-aligned multilayer network to calculate the degree. This results in tens of thousands of missing nodes being added to the layers, which ultimately leads to higher memory usage and causes the process to terminate for larger networks. An additional example of memory consumption in multiNetX for a multilayer network that is not node-aligned can be found in Appendix D.
8 Related work

In a previous study, various multilayer network software were compared, with a particular emphasis on their visualisation capabilities. A conference proceeding article from 2017 analysed the visualisation features of muxViz, Pymnet, multiNetX and MultiNets \cite{32}. MultiNets is a JavaScript library designed for multilayer network visualisation \cite{33}, which was not analysed in this thesis (not to be confused with Multinet). However, this report considered all other libraries examined in the survey. Although this thesis did not primarily focus on multilayer network visualisation, the article from 2017 offers a comprehensive analysis of the current challenges and potential research directions in this field.
9 Conclusion

This thesis conducted a comparison of four multilayer network software based on their performance, design, and capabilities. The analysis involved evaluating temporal multilayer networks, which influenced the selection of metrics used for comparison.

Pymnet and Multinet were applicable in all use cases, but muxViz performed the best when the scalability of all software was tested for the aggregation operation. multiNetX extends the Python library NetworkX for multilayer network analysis and was lacking in features compared to the other software. However, multiNetX was still analysed in this work to determine how custom implementations could extend its capabilities if needed, and how those custom implementations could be used for analysing temporal networks along with the other three software.

When testing the scalability of calculating the degree of the nodes in a temporal network, weaknesses in the software were discovered. Both muxViz and multiNetX experienced difficulties when handling larger multilayer networks while attempting the degree calculation scalability test. multiNetX could not complete the calculation due to exceeded memory usage since multilayer networks in multiNetX must be node-aligned for degree measurements. While muxViz encountered internal library errors. Furthermore, as the largest network sizes were used for testing, the time taken to perform these operations using Multinet significantly increased. Upon analysis, it was found that the version of Multinet used during testing was ported from C++ to Python using PYBIND11, which may have resulted in additional overhead in memory management.

A recognised weakness with the work is that it relied on a single dataset to assess all libraries’ performance and practical applications. This constraint prevented the evaluation of other metrics that could have unlocked the software’s full potential, as they were tailored for analysing different types of multilayer networks, such as multiplex networks.
10 Future work

Considering that this project compared multilayer network software using a single dataset and a generalised implementation method, the fairness and accuracy of the analysis can be enhanced.

10.1 Incorporating additional datasets to the analysis

In order to fully assess the capabilities of the software in importing various types of multilayer networks, it would be beneficial to test it with additional datasets. For instance, a multilayer network model with two dimensions could be utilised to expand the scope of the software’s capabilities. When selecting a dataset, it is important to also consider the multilayer networks that the chosen software supports. This can help ensure consistency across all software used when creating multilayer networks from a model.

10.2 Expanding the scope of the scalability testing

This study compared multilayer network operations for all software, making finding more operations to test difficult. Instead, to provide a more detailed analysis, the comparison can be broken down into separate analyses for each subset of the software, where appropriate. This would increase the number of operations compared to the investigation.

10.3 Adapting the implementation design

To ensure a fair comparison of the selected software, it may be beneficial to limit testing to its native programming language. This can prevent any potential overhead during data collection, as seen in this report as a porting of one of the libraries was used instead of its native implementation.
REFERENCES

References


REFERENCES


REFERENCES


Appendix A

./dataset/

1900s
- blocks_fields.txt
- blocks_fields.txt_names
- blocks_level_0.txt
- blocks_level_1.txt
- blocks_level_2.txt
- blocks_level_3.txt
- blocks_level_4.txt
- edgelist_level_0.txt
- edgelist_level_1.txt
- edgelist_level_2.txt
- edgelist_level_3.txt
- edgelist_level_4.txt

1901s
- blocks_fields.txt
- blocks_fields.txt_names
- blocks_level_0.txt
- blocks_level_1.txt
- blocks_level_2.txt
- blocks_level_3.txt
- blocks_level_4.txt
- blocks_level_5.txt
- edgelist_level_0.txt
- edgelist_level_1.txt
- edgelist_level_2.txt
- edgelist_level_3.txt
- edgelist_level_4.txt

2008-2013
- blocks_fields.txt
- blocks_fields.txt_names
- blocks_level_0.txt
- blocks_level_1.txt
- blocks_level_2.txt
- blocks_level_3.txt
- blocks_level_4.txt
- blocks_level_5.txt
- blocks_level_6.txt
- blocks_level_7.txt
- blocks_level_8.txt
- blocks_level_9.txt
- edgelist_level_0.txt
- edgelist_level_1.txt
- edgelist_level_2.txt
- edgelist_level_3.txt
- edgelist_level_4.txt
- edgelist_level_5.txt
- edgelist_level_6.txt
- edgelist_level_7.txt
- edgelist_level_8.txt
- edgelist_level_9.txt

105 directories, 2100 files

Appendix A continues on the next page
<table>
<thead>
<tr>
<th>Naming Convention</th>
<th>Content</th>
<th>Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocks fields.txt</td>
<td>Active journals categorised to their assigned fields for the current time window directory.</td>
<td>This file is ignored for this thesis.</td>
</tr>
<tr>
<td>blocks fields.txt.names</td>
<td>The names of the fields for the current time window directory, for example biology and medicine. The number and types of fields vary based on the current time window.</td>
<td>This file is ignored for this thesis.</td>
</tr>
<tr>
<td>blocks_level_0.txt</td>
<td>Active journals/nodes for the current time window directory. There is one line for every journal in the file and the content of every line is the title of the journal.</td>
<td>Higher levels such as &quot;blocks_level_1.txt&quot; are ignored for this thesis.</td>
</tr>
<tr>
<td>edgelist_level_0.txt</td>
<td>Edges for the current time window in the format; source target weight. There is one line for every edge in the file and the numerical value weight is ignored for this thesis. The values of source and target are indices using the line number from &quot;blocks_level_0.txt&quot; for the current time window directory (indices start from zero).</td>
<td>Higher levels such as &quot;edgelist_level_1.txt&quot; are ignored for this thesis.</td>
</tr>
</tbody>
</table>
Appendix B

The layer difference of the layers 1900s and 1910s visualised using Multinet. Nodes that have a higher number of incoming edges are displayed with a larger size.
Appendix C

Query 3: Find the top 100 influencing journals between 1900-2010.

<table>
<thead>
<tr>
<th>Journal</th>
<th>Citations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Science,20855</td>
<td></td>
</tr>
<tr>
<td>Nature,2011</td>
<td></td>
</tr>
<tr>
<td>Nobata,17717</td>
<td></td>
</tr>
<tr>
<td>Lancet,10024</td>
<td></td>
</tr>
<tr>
<td>N._Engl._J._Med.,9838</td>
<td></td>
</tr>
<tr>
<td>Phys._Rev._Lett.,9103</td>
<td></td>
</tr>
<tr>
<td>Ann._NY_Acad._Sci.,8924</td>
<td></td>
</tr>
<tr>
<td>J._Biol._Chem.,8815</td>
<td></td>
</tr>
<tr>
<td>JAMA-J._Am._Med._Assoc.,8640</td>
<td></td>
</tr>
<tr>
<td>Br._Med., J.,7769</td>
<td></td>
</tr>
<tr>
<td>Biochem._Biophys._Res._Commun.,7570</td>
<td></td>
</tr>
<tr>
<td>J._Am._Chem._Soc.,7556</td>
<td></td>
</tr>
<tr>
<td>J._Appl._Phys.,7508</td>
<td></td>
</tr>
<tr>
<td>Appl._Phys._Lett.,7118</td>
<td></td>
</tr>
<tr>
<td>Anal._Chem.,6934</td>
<td></td>
</tr>
<tr>
<td>Sci._Am.,6848</td>
<td></td>
</tr>
<tr>
<td>Cancer_Res.,6725</td>
<td></td>
</tr>
<tr>
<td>J._Clin._Invest.,6686</td>
<td></td>
</tr>
<tr>
<td>Cell,6672</td>
<td></td>
</tr>
<tr>
<td>PFEBS_Lett.,6622</td>
<td></td>
</tr>
<tr>
<td>Circulation,6581</td>
<td></td>
</tr>
<tr>
<td>Nucleic_Acids_Res.,6478</td>
<td></td>
</tr>
<tr>
<td>Ann._Intern._Med.,6440</td>
<td></td>
</tr>
<tr>
<td>Biochem._J.,6438</td>
<td></td>
</tr>
<tr>
<td>Proc._IEEE,6434</td>
<td></td>
</tr>
<tr>
<td>Biochemistry,6378</td>
<td></td>
</tr>
<tr>
<td>Appl._Optics,6233</td>
<td></td>
</tr>
<tr>
<td>Pediatrics,6098</td>
<td></td>
</tr>
<tr>
<td>Phys._Rev._B.,5852</td>
<td></td>
</tr>
<tr>
<td>J._Immunol.,5799</td>
<td></td>
</tr>
<tr>
<td>J._Mol._Biol.,5722</td>
<td></td>
</tr>
<tr>
<td>Cancer,5716</td>
<td></td>
</tr>
<tr>
<td>Blood,5694</td>
<td></td>
</tr>
<tr>
<td>Neurology,5665</td>
<td></td>
</tr>
<tr>
<td>Biophys._J.,5657</td>
<td></td>
</tr>
<tr>
<td>Arch._Intern._Med.,5625</td>
<td></td>
</tr>
<tr>
<td>Phys._Rev._B.,5620</td>
<td></td>
</tr>
<tr>
<td>J._Neurosci.,5584</td>
<td></td>
</tr>
<tr>
<td>J._Clin._Endocrinol._Metab.,5540</td>
<td></td>
</tr>
<tr>
<td>J._Radiol._Cancer_Inst.,5509</td>
<td></td>
</tr>
<tr>
<td>J._Cell_Biol.,5492</td>
<td></td>
</tr>
<tr>
<td>Brain_Res.,5479</td>
<td></td>
</tr>
<tr>
<td>Nature_Genet.,5433</td>
<td></td>
</tr>
<tr>
<td>Anal._Biochem.,5412</td>
<td></td>
</tr>
<tr>
<td>Rev._Sci._Instrum.,5406</td>
<td></td>
</tr>
<tr>
<td>Arch._Biochem._Biophys.,5396</td>
<td></td>
</tr>
<tr>
<td>J._Chem._Phys.,5340</td>
<td></td>
</tr>
<tr>
<td>Am._J._Clin._Nutr.,5336</td>
<td></td>
</tr>
<tr>
<td>J._Pharmacol._Exp._Ther.,5305</td>
<td></td>
</tr>
<tr>
<td>J._Physiol._London,5302</td>
<td></td>
</tr>
<tr>
<td>Gastroenterology,5298</td>
<td></td>
</tr>
<tr>
<td>Clin._Chem.,5292</td>
<td></td>
</tr>
<tr>
<td>Endocrinology,5264</td>
<td></td>
</tr>
<tr>
<td>Physiol._Rev.,5263</td>
<td></td>
</tr>
<tr>
<td>J._Theor._Biol.,5220</td>
<td></td>
</tr>
<tr>
<td>Life_Sci.,5214</td>
<td></td>
</tr>
<tr>
<td>Nat._Med.,5198</td>
<td></td>
</tr>
<tr>
<td>Br._J._Cancer,5198</td>
<td></td>
</tr>
<tr>
<td>Am._J._Epidemiol.,5177</td>
<td></td>
</tr>
<tr>
<td>J._Nutr.,5175</td>
<td></td>
</tr>
<tr>
<td>Faseb_J.,5167</td>
<td></td>
</tr>
<tr>
<td>Biochem._Pharmacol.,5159</td>
<td></td>
</tr>
<tr>
<td>Nat._Biotechnol.,5148</td>
<td></td>
</tr>
<tr>
<td>Am._J._Pathol.,5141</td>
<td></td>
</tr>
<tr>
<td>J._Exp._Med.,5113</td>
<td></td>
</tr>
<tr>
<td>J._Appl._Physiol.,5069</td>
<td></td>
</tr>
<tr>
<td>J._Pediatr.,5068</td>
<td></td>
</tr>
<tr>
<td>Appl._Environ._Microbiol.,5057</td>
<td></td>
</tr>
<tr>
<td>Int._J._Cancer,5002</td>
<td></td>
</tr>
<tr>
<td>Am._J._Hum._Genet.,4998</td>
<td></td>
</tr>
<tr>
<td>Commun._ACM,4983</td>
<td></td>
</tr>
<tr>
<td>Am._J._Obstet._Gynecol.,4956</td>
<td></td>
</tr>
<tr>
<td>Embo_J.,4909</td>
<td></td>
</tr>
<tr>
<td>Radiology,4893</td>
<td></td>
</tr>
<tr>
<td>Chem._Rev.,4881</td>
<td></td>
</tr>
<tr>
<td>J._Infect._Dis.,4852</td>
<td></td>
</tr>
<tr>
<td>Angew.<em>Chem.</em>-Int._Edit.,4851</td>
<td></td>
</tr>
<tr>
<td>Exp._Cell_Res.,4826</td>
<td></td>
</tr>
<tr>
<td>Neurosci._Lett.,4821</td>
<td></td>
</tr>
<tr>
<td>J._Agric._Food_Chem.,4818</td>
<td></td>
</tr>
<tr>
<td>Annu._Rev._Biochem.,4814</td>
<td></td>
</tr>
<tr>
<td>Eur._J._Biochem.,4755</td>
<td></td>
</tr>
<tr>
<td>J._Neurochem.,4752</td>
<td></td>
</tr>
<tr>
<td>Can._Med._Assoc.,J.,4750</td>
<td></td>
</tr>
<tr>
<td>Electron_Lett.,4736</td>
<td></td>
</tr>
<tr>
<td>Environ._Health_Perspect.,4689</td>
<td></td>
</tr>
<tr>
<td>Am._J._Psychiat.,4688</td>
<td></td>
</tr>
<tr>
<td>J._Bacteriol.,4680</td>
<td></td>
</tr>
<tr>
<td>Philos._Trans._R._Soc._Biol._Sci.,4678</td>
<td></td>
</tr>
<tr>
<td>Environ._Sci._Technol.,4673</td>
<td></td>
</tr>
<tr>
<td>Clin._Chim._Acta,4610</td>
<td></td>
</tr>
<tr>
<td>Chest,4572</td>
<td></td>
</tr>
<tr>
<td>Circ._Res.,4546</td>
<td></td>
</tr>
<tr>
<td>Diabetes,4520</td>
<td></td>
</tr>
<tr>
<td>J._Urol.,4490</td>
<td></td>
</tr>
<tr>
<td>J._Virol.,4480</td>
<td></td>
</tr>
<tr>
<td>J._Cell_Sci.,4474</td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

Memory consumption in mebibytes over time measured using memory profiler when performing query executions in multiNetX for the time windows 1900-1970s. The multilayer network was not node-aligned for this execution reducing the peak memory usage count compared to Figure 17.