Final Report: Advanced Bioinformatic Algorithms

NMVis

A Web-Based Network Motif Visualization Tool

Chase Meyer

March 17th 2019

Introduction:

A network motif is a unique pattern in a network that has a statistically significant frequency of occurrence. It is a subgraph of a network that, in a biological context, can indicate an interaction between biological molecules that is more significant than the other interactions in the network. In bioinformatics, the nodes in the network represent biomolecules such as proteins, DNA, RNA, and metabolites. An edge between these biomolecules represents an interaction. Network motifs are important for biological researchers to use to discover a wide range of useful information. For example, network motifs could be used find proteins essential to diseases or genes involved in complex biological interactions. As of now there are a very limited number of tools used analyze and visualize network motifs, and the ones currently available are dated and have restrictions to certain machine architectures. The current standard in network motif detection is FANMOD, a tool for fast network motif detection [13]. FANMOD is great for detecting the motifs present in a graph, however it is not capable of providing the nodes present in each instance of the motif and provides no visual representation of the motifs apart an image of the general motif structure. Other tools are capable of visualizing the network motifs such as MAVisto and mDraw, however these tools are extremely slow at processing and visualizing the motifs in networks of any scale and for all but the smallest of network motifs [13, 9]. This is because as the size the network increases or the size of the motif to be detected increases, there is an exponential increase in the number of possible subgraphs. Both of these visualization tools are also very dated and are desktop applications, limiting their portability and usefulness across various system architectures. The primary motivation of this project is to create an updated visualization tool that is easy for biological researchers to understand and use, is portable, and is capable of being scaled to fit the needs of large networks and motif sizes. Creating this tool in a web-based environment will support these motivations in addition to making it easily accessible and thus adoptable by biological researchers interested in trying a new tool.

The primary objective of this tool is to create an interactive environment for viewing the motifs within a network, however a secondary goal of the project is to provide the groundwork for later integrating network motif detection capabilities. For this reason, much of the work completed throughout this project takes into consideration the future integration of NemoLib, a library for efficient detection of network motifs [11].

Program description:

NMVis is a web-based project for interactive visualization of network motifs. The tool uses user provided files to render input graphs and the motifs present in them. What exactly is meant by interactive? The tool is interactive because the graphs visualized respond to manipulation by the client though a mouse or trackpad. This is particularly useful for visualizing network motifs since their isomorphic representations can have a number of distinct viewing angles that can be difficult to envision from a still image of just one. This is especially true of larger motif sizes. For example, figure 1 shows four different viewing angles for the same size 8 motif that all look drastically different. Using this tool allows the user to drag the nodes/edges around to see a motif from any angle. When viewing a particular motif instance in a complex and busy graph, this is an invaluable functionality.

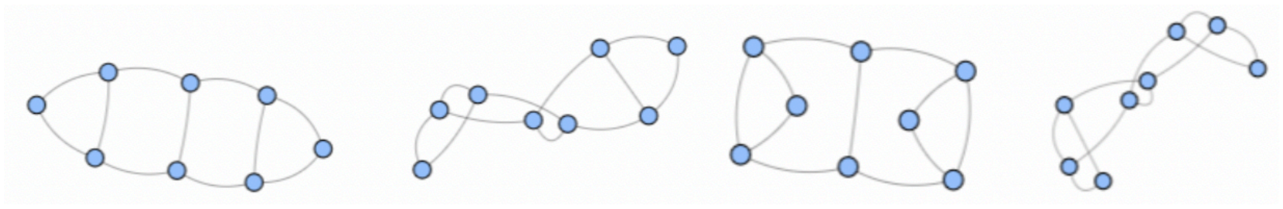
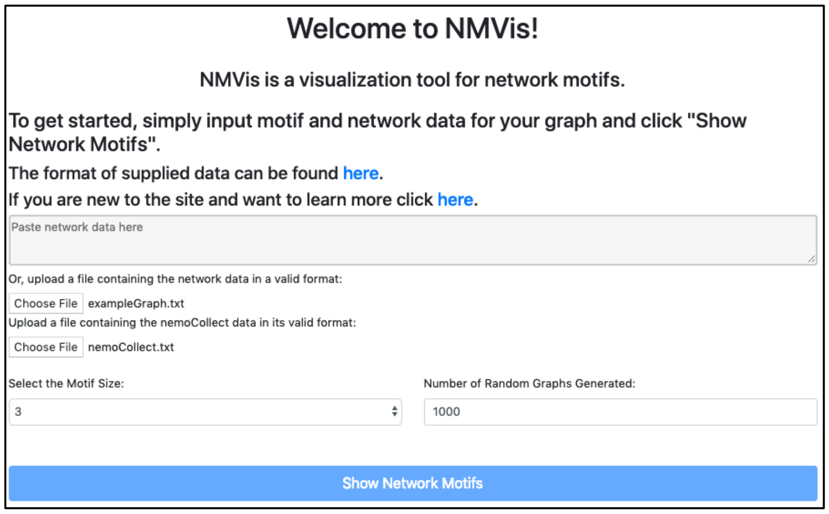


Figure 1: These images show the same size 8 motif from four different viewing angles and helps illustrate the usefulness of an interactive graph. These images were generated using NMVis.

As for the functionality currently present in NMVis, the tool is capable of a variety of tasks including: simple graph visualization (no motifs), visualizing the motifs present in the nemoCollect file, and highlighting/focusing on each motif instances in the full graph. A user begins on the home page (Figure 2). To display an input graph, simply paste the graph into the input box or attach the graph file without attaching a nemoCollect file. To visualize the motifs, their counts (number present in the file), and id, include the nemoCollect file and indicate the motif size. The nemoCollect file contains all motifs found through enumeration, with the IDs of each motif instance and the nodes present in each motif. The format of this data can be found on the site.



Welcome to NMVis!

NMVis is a visualization tool for network motifs.

To get started, simply input motif and network data for your graph and click "Show Network Motifs".

The format of supplied data can be found here.

If you are new to the site and want to learn more click here.

Paste network data here

Or, upload a file containing the network data in a valid format:

Choose File exampleGraph.txt

Upload a file containing the nemoCollect data in its valid format:

Choose File nemoCollect.txt

Select the Motif Size: 3

Number of Random Graphs Generated: 1000

Show Network Motifs

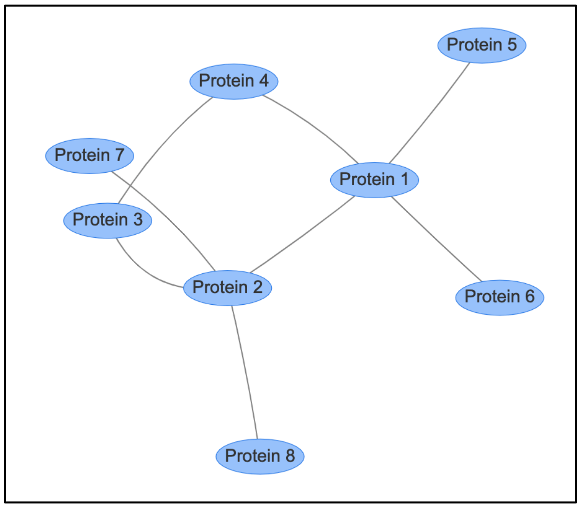
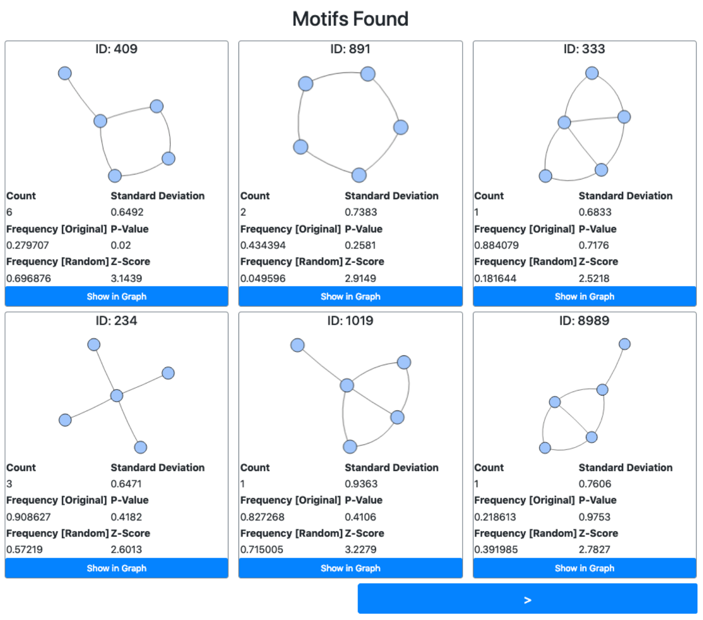


Figure 2 (left): View of the starting page of the NMVis website with the form required by the client. As of now the number of random graphs field is not actually displayed on the site. It is shown here to illustrate the sites appearance with the implementation the of NemoLib functionalities.

Figure 3 (right): This image shows a simple interactive graph visualized with NMVis. The graph uses the underlying vis.js library for rendering.

The resulting motifs will be displayed in a table of cards below the graph visualization, as shown in figure 4. As described, each motif graph can be interacted within the overall graph. It is important to note that while each motif card currently includes a space for standard deviation, frequency, p-value, and z-score, this information not be available until the integration of NemoLib and is only shown in figure 4 to illustrate upcoming functionality. With the current version of NMVis, unavailable values show “N/A”. A specific motif instance can be highlighted in the network by clicking on the “Show in Graph” button.



ID: 409

Count	Standard Deviation
6	0.6492
Frequency [Original]	**P-Value**
0.279707	0.02
Frequency [Random]	**Z-Score**
0.696876	3.1439

ID: 891

Count	Standard Deviation
2	0.7383
Frequency [Original]	**P-Value**
0.434394	0.2581
Frequency [Random]	**Z-Score**
0.049596	2.9149

ID: 333

Count	Standard Deviation
1	0.6833
Frequency [Original]	**P-Value**
0.884079	0.7176
Frequency [Random]	**Z-Score**
0.181644	2.5218

ID: 234

Count	Standard Deviation
3	0.6471
Frequency [Original]	**P-Value**
0.908627	0.4182
Frequency [Random]	**Z-Score**
0.57219	2.6013

ID: 1019

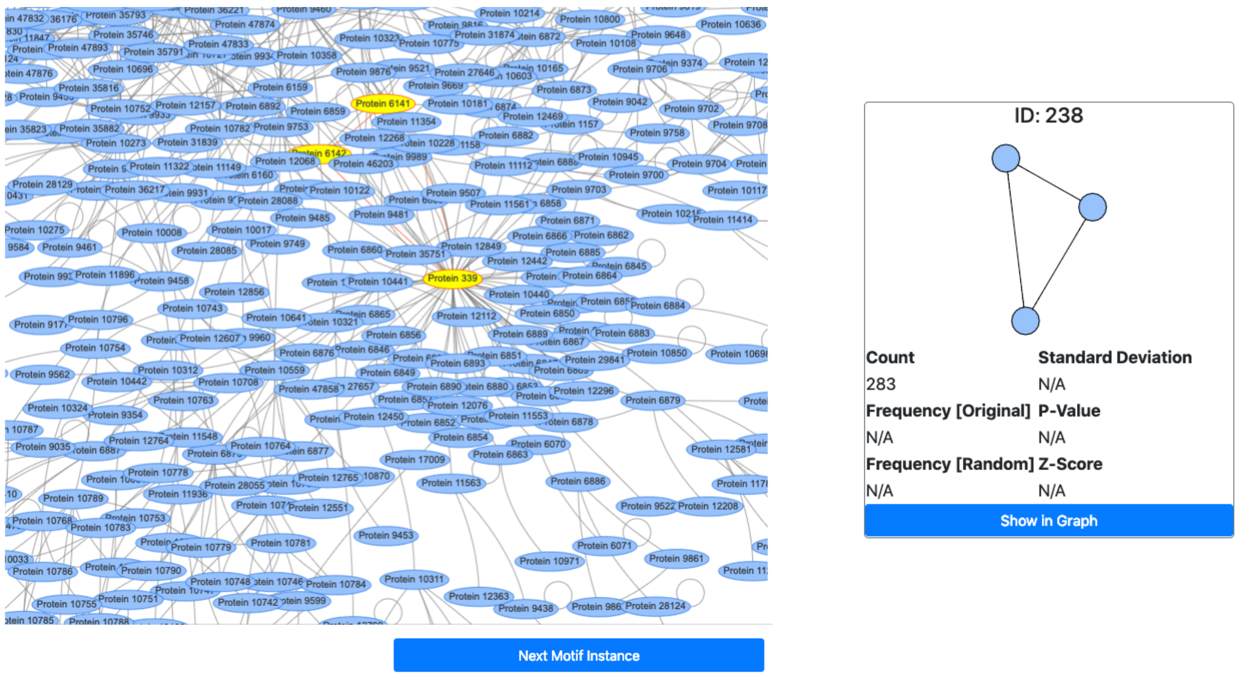
Count	Standard Deviation
1	0.9363
Frequency [Original]	**P-Value**
0.827268	0.4106
Frequency [Random]	**Z-Score**
0.715005	3.2279

ID: 8989

Count	Standard Deviation
1	0.7606
Frequency [Original]	**P-Value**
0.218613	0.9753
Frequency [Random]	**Z-Score**
0.391985	2.7827

Figure 4: This shows a table of possible size 5 motifs as they will be displayed using the NMVis tool after the upcoming implementation of NemoLib. The cards currently show N/A for the values unavailable.

This is shown in figure 5 for a large graph. Highlighting a motif instance will shift the view such that all motif nodes are in view and will change the color of the edges and node outlines to red and fill color of the nodes to yellow. Clicking on the “Next Motif Instance” button will move to the next motif instance in the graph. This button and the buttons below the cards will only be displayed if there are multiple motif instances or card pages, respectively.



ID: 238

Count	**Standard Deviation**
283	N/A
Frequency [Original]	**P-Value**
N/A	N/A
Frequency [Random]	**Z-Score**
N/A	N/A

Figure 5: This figure shows the highlighted view of specific motif instance in the network for the motif in the card shown to the right. The left image also helps depict the tools usefulness for large finding motifs in large networks.

The project follows a client-server style architecture, with a front-end written in HTML, CSS, and JavaScript and a back-end written in Java. JavaScript visualization library vis.js is used to render all of the graphs within NMVis. Notably, the current state of the project technically does not require the Java back-end since it is currently only parsing and processing the data in a format that can be used by the vis.js library. In the initial implementation of the project, code was written to do most of this in JavaScript on the front-end. This did work; however, it was extremely slow at processing the data for graphs of any scale due to the single-threaded nature of browser processing. This is not an ideal solution and contrasts my motivation of developing a scalable application. The front-end also already has a fairly large amount of work to do in order to render the graphs and process the interactions with the graph components. For these reasons, and because of a desire to extend the project to include NemoLib, I pushed the processing task to the back-end. This dramatically improved the performance of the system.

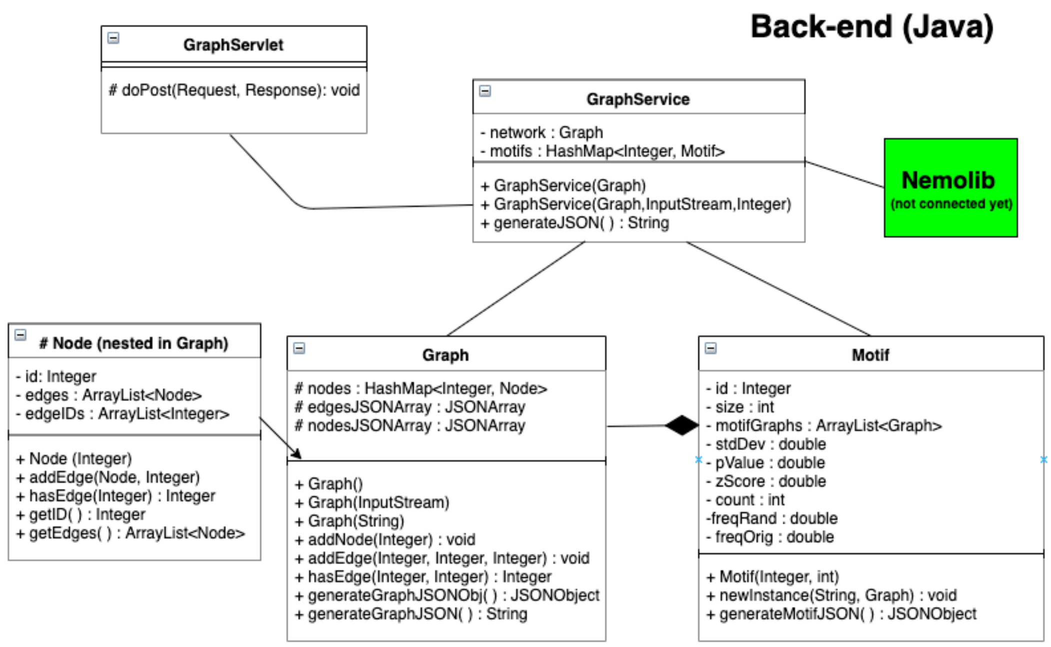
Communication between the front-end and the back-end is done through a single RESTful API. This will ideally be split up into multiple different APIs with the integration of NemoLib in order to facilitate the asynchronous processing of the motifs on the back-end while the front-end renders and displays the graph and any nemoCollect data directly input by the user (what the system is currently doing).
The current code has been written with this in mind, providing a level of modularity in the current classes to facilitate this addition. The REST API currently in place is an HTTP POST request. While there is no creation of lasting resources with this request, a POST request was chosen because, often large, files are passed between the client and the server, after they are input into a form on the site. Since the request does not actually change the state of the server it is idempotent, contrary to the typical POST request.

NMVis is hosted at www.nmvis.azurewebsites.net as an Azure Web App Service. Hosting the system on a cloud platform helps to keep it portable and makes scaling easy. If usage of the service is too much for the resources currently allocated to the service, more memory or processing power can be added with a few clicks.

Considerations:

When designing the application, it was important to consider the capabilities of all system components to verify the usability and performance of the tool. First and foremost, this meant designing the visualization tool in a way that would not hamper its performance in the average user's browser and with its available memory. It also meant making sure the tool's fundamental functionality (displaying the motif cards) would work on the majority of client's browsers. These are important factors to consider for this tool in particular because of the substantial amount of memory and CPU power needed to render complex graphs with JavaScript in a browser. To help mitigate unnecessary rendering, NMVis gives the user the option of visualizing the full network but requires them to click a button to verify that they want to do so. For large graphs (generally around 1000 nodes), rendering can be fairly time-consuming on older machines, therefore, for users that don't need this, it is unnecessary to bog down the system rendering unneeded graphs. Also, as previously discussed, by processing the graphs on the back-end, the user's browser is absolved of this task, increasing usability in slower client systems. While it likely would not have a huge impact on performance, the front-end is coded in standard JavaScript instead of using a framework to make sure performance is as optimal as possible and memory use is not too high.

Another consideration for the system was the size of the graph allowed for input. The front-end places a 50000 byte (50 KB) limit on the file size of an input graph, as graphs larger than this will have poor rendering performance. A graph file of 50KB amounts to roughly 3000 edges. This restriction will likely be modified when network motif detection functionality is later implemented with NemoLib, so that the motifs can be determined by the back-end but visualization will be limited. By placing this restriction on the front-end (in the validateForm JavaScript function), the client can technically override this functionality by editing the in-browser files if they have the desire to, though it will likely not produce a usable visualization. Lastly, the site only allows a user to attach text documents and limits the motif size to 8, the standard for most motif detection programs and the NemoLib library. The site also maintains a valid https certificate in order to protect itself against possible security threats that would be applicable to the particular system.



Back-end (Java)
GraphServlet
# doPost(Request, Response): void
GraphService
- network : Graph
- motifs : HashMap<Integer, Motif>
+ GraphService(Graph)
+ GraphService(Graph,InputStream,Integer)
+ generateJSON( ) : String
Nemolib
(not connected yet)
# Node (nested in Graph)
- id: Integer
- edges : ArrayList<Node>
- edgeIDs : ArrayList<Integer>
+ Node (Integer)
+ addEdge(Node, Integer)
+ hasEdge(Integer) : Integer
+ getID( ) : Integer
+ getEdges( ) : ArrayList<Node>
Graph
# nodes : HashMap<Integer, Node>
# edgesJSONArray : JSONArray
# nodesJSONArray : JSONArray
+ Graph()
+ Graph(InputStream)
+ Graph(String)
+ addNode(Integer) : void
+ addEdge(Integer, Integer, Integer) : void
+ hasEdge(Integer, Integer) : Integer
+ generateGraphJSONObj( ) : JSONObject
+ generateGraphJSON( ) : String
Motif
- id : Integer
- size : int
- motifGraphs : ArrayList<Graph>
- stdDev : double
- pValue : double
- zScore : double
- count : int
-freqRand : double
- freqOrig : double
+ Motif(Integer, int)
+ newInstance(String, Graph) : void
+ generateMotifJSON( ) : JSONObject

Figure 6: This figure shows the UML class diagram for Java back-end in the current system.

Methods, algorithms and tools:

Back-end:

The primary role of the Java back-end at this time is to process the user-supplied graph and nemoCollect data from the POST request, and convert these into JavaScript Object Notation (JSON) format so that it can be used by vis.js and other JavaScript functions for a number of tasks. Most of the back-end functionality is fairly straightforward, however I will explain some components. The components and their relationships are further explained by the UML in figure 6.

Java Servlets are used on the back-end to facilitate client-server communication. In the implementation, a GraphServlet class extends the HttpServlet interface overriding the doPost method. The doPost method gets the “motifSize” and file parameters from the HttpServletRequest object passed from the form on the front-end. A Graph object is created to handle the logic associated with parsing the file or text input containing the graph file and a GraphService object is created to manage the remaining business logic of the back-end, namely the processing of the motif data. The GraphService object is passed the created Graph object and nemoCollect file, which it uses to find all of the edges for each motif instance. A Motif class is also been created to process and store data for each motif, including all of its instances, which are stored as separate Graph objects in an ArrayList. Classes on the back-end were designed to allow for easy integration with NemoLib in the next version of NMVis. The Motif and Graph classes each contain methods for generating a JSONObject representation of their data. These functions are called by the generateJSON method in the GraphService class to combine the processed data into a single JSON object that gets converted into a string and later used as the response for the GraphServlet’s doPost.

In this JSON object, a graph object with “nodes” and “edges” attributes is used to represent the graph, a separate JSONArray named “motif” contains objects representing the data for each motif. Each of these motifs will have a separate JSONArray with the matching motif instance subgraphs.

Front-end:

The front-end is fairly straightforward as well though I will go into some details regarding the use of vis.js. The vis.js library requires data in a specific format in order to render network visualizations, hence the need to process the data. Graph data is supplied to the vis.js library in JSON format, nodes must have an “id” attribute with a unique value, and edges must have “to” and “from” attributes. I have also assigned a unique “id” attribute to each edge in the graph on the back-end. The “id” attribute (for both the edges and the nodes) will be the same in the JSON object for the full graph as it is in the JSON object for each motif instance subgraph. This unique id is necessary for the identification of nodes and edges to be highlighted in the main network. Before a graph is being created with vis.js, all of the node and edge data is first stored in a DataSet object, also from the vis.js library. This DataSet object, a designated HTML div, and an options JavaScript object are then used to generate the network. In order to change the color of the nodes and edges in the visualization dynamically, the DataSet nodes and edges are altered in the DataSet object, causing a redraw of the network with the updates.

On the NMVis site, the user never navigates away from the home page of the site, the page is instead dynamically updated by changing the CSS “style.display” attribute from ‘none’ to a visible form (usually ‘block’). The various components in the page are updated with relevant information before displaying the hidden elements to provide a streamlined experience for the user.

Data, Testing and Results:

Most of the testing for this program involved manually performing unit testing on each component of the application since the output of the tool are visualizations. To test the validity of the graphs themselves, I started by initially testing a small graph as input. Once this was verified I moved on to testing larger graphs before testing the visualization results of the motifs. Each individual component was tested completely before moving onto other components. To test the accuracy of the POST request and JSON data passed with the request, swagger.io was used with both small and large test files. Figure 5 shown previously shows the results of testing a large graph.

The testing for NMVis did not include testing using Internet Explorer and the application is only guaranteed to supports current browser platform versions. Browser compatibility testing was completed using Safari, Google Chrome, Microsoft Edge, and Mozilla Firefox. These browsers all produced adequate testing results and have similar operating conditions.

Some known limitations of the program include defects in the format and input of data. This will cause affected components to not be displayed in the output. This could be mitigated on the back-end, however, it would cause a decrease in efficiency to check for these inaccuracies. Since the input nemo collect file will likely be computer generated, the program assumes it will not have any fatal errors. One rather large limitation of NMVis is the current need for the network motif collection data to be given by the user. This data can be difficult to collect and, in the case of NemoLib is currently not being recorded due to performance and storage needs. For large graphs, the text that would need to be recorded exceeds available memory for most systems, causing a drastic impact on performance. This problem is currently being addressed in NemoLib.

Conclusion:

In order for biologists to fully understand and utilize network motif data, it is necessary to obtain and visualize the resulting motif instances so that they can more easily interpret motif data and interaction of the present biomolecules. NMVis is a visualization tool designed to do just that. The tool is a much-needed alternative to dated desktop applications that have previously been used to perform this task on small graphs, and provides a viable method for visualizing larger graphs which could not be easily visualized by previous programs.

Future work for NMVis will start with the integration of NemoLib, a library for network motif detection. Some of this functionality has already been implemented in the application which should make connecting the library on the backend a little easier. The integration of NemoLib will allow the tool to collect the network motif instance data using the NemoLib library. This is the data currently provided by the user to the program. Other areas for future work on the project could involve extending the tool’s functionalities to work with directed graphs, further improving performance, or combining the tool with database systems of specific projects to optimize performance and usability.

References:

[1]

“Vis.js.” [Online]. Available: http://visjs.org/docs/network/

[2]

“Create a Hello World web app for Azure using IntelliJ.” [Online]. Available:

https://docs.microsoft.com/en-us/java/azure/intellij/azure-toolkit-for-intellij-create-hello-world-web-app?view=azure-java-stable

[3]

“Java Server-side Programming.” [Online]. Available:

https://www.ntu.edu.sg/home/ehchua/programming/java/JSPByExample.html. [Accessed: 3/4/19]

[4]

“Java SE 8: Creating a Web App with Bootstrap and Tomcat Embedded.” [Online]. Available:

https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/basic_app_embedded_tomcat/basic_app-tomcat-embedded.html

[5]

F. Schreiber and H. Schwbbermeyer, “MAVisto: a tool for the exploration of network motifs,” vol. 21, no. 17, pp. 3572–3574, 2005.

[6]

S. A. Thomas, *Data visualization with JavaScript*. San Francisco, CA: San Francisco, CA : No Starch Press, 2015.

[7]

A. R. Dalby, A. Masoudi-Nejad, M. Ansariola, M. K. Razaghi, A. Salehzadeh-Yazdi, and S. Khakabimamaghani, “CytoKavosh: A Cytoscape Plug-In for Finding Network Motifs in Large Biological Networks (Cytoscape Plug-In for Finding Network Motifs),” vol. 7, no. 8, p. e43287, 2012.

[8]

"Java(TM) EE 8 Specification APIs." [Online]. Available:

https://docs.oracle.com/javaee/8/api/javax/servlet/http/HttpServletRequest.html. [Accessed: 3/7/19]

[9]

H. Schwbbermeyer and R. Wnschiers, "MAVisto: a tool for biological network motif analysis," vol. 804, p. 263, 2012.

[10]

B. H. Junker, F. Schreiber, B. H. Junker, and F. Schreiber, *Analysis of biological networks*. Hoboken, N.J.: Hoboken, N.J. : Wiley-Interscience, 2008.

[11]

A. Andersen and W. Kim, "NemoLib: A java library for efficient network motif detection," vol. 10330. pp. 403–407, 2017.

[12]

"w3schools.com." [Online]. Available: https://www.w3schools.com/. [Accessed: 3/15/19]

[13]

S. Wernicke and F. Rasche, "FANMOD: a tool for fast network motif detection," vol. 22, no. 9, pp. 1152–1153, 2006.