University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Biological Network Analysis and Subnetwork Extraction Tool (Biological Net Extranalysis)

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Danielle M. Rodriguez

June 2017

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Biological Network Analysis and Subnetwork Extraction Tool (Biological Net Extranalysis)" prepared and submitted by Danielle M. Rodriguez in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Marvin John C. Ignacio, M.Sc
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Vincent Peter C. Magboo, M.D., M.Sc.	_____	_____
2. Richard Bryann L. Chua, M.Sc.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

Research has presented that when dealing with complex protein chains of various diseases, there is a large amount of proteins that interact with each other in different ways. Due to the complexity of the control mechanisms involved, and the large number of possible interactions, there is a great need for computer-assisted tools.

The Biological Net Extranalysis is a tool that helps the user visualize a graph, extract a subnetwork containing the relationships that are significant in the network, and perform network analysis on the graph.

Keywords: biological networks, graph theory, subnetwork, graph centrality

Contents

List of Figures

I. Introduction

A. Background of the Study

Being able to represent cellular networks with the use of graphs and using these graphs in order to compute quantitative measures describing their topology are helpful in the search for more systems-level insights regarding cellular regulation. The construction and analysis of such graphs and the network they represent have become major research topics in systems biology [1]. Various aspects of these networks have been analyzed including the inference of cellular networks from gene expression, network alignments and other related strategies [2].

Due to the complexity of the control mechanisms involved, and the large number of possible interactions, there is a great need for computer-assisted tools to manage, query and interpret the experimental observations with formal network models [3]. With the advent of high-throughput techniques for network analysis, large-scale identification of components (genes, RNAs, and proteins), their expression patterns, and their biochemical and genetic interactions have been possible. The data generated from the use of such techniques in various studies can provide valuable information about the functions of individual components and unexpected relationships between components and cellular processes [4].

Looking at different network properties can provide valuable insight into the internal organization of a biological network, the repartition of molecules among cellular processes, as well as the evolutionary constraints that have shaped an organisms protein, metabolic or regulatory network into a functional, feasible structure [5]. The centralities of a node are used to measure its contribution to the communication between other nodes [6]. Zhao et al., for example, found that metabolic functions were carried out in an ordered and modular way and the topological features of metabolic network could provide a functional implication [7].

Another important problem in network analysis is the extraction of a subnetwork that may suggest a process of interest that is composed of a connections of implicated proteins [8]. This is useful in being able to find any coherence between the aforementioned proteins and to draw any further conclusions regarding their function in the network.

The outcomes of this work form part of the "Mathematical Modelling of Biochemical Pathways" project, an interdisciplinary effort being carried out in partnership with the Mathematical and Statistical Modeling Unit under the Center for Natural Sciences and Environmental Research (CENSER) of De La Salle University. The mathematical models that will be established in this project will try to explain the effect of a natural product (as drug target) on three different cancer diseases namely breast cancer, colon cancer and leukemia.

The input to be used in this work will come from another process that will take a pathway from the Kyoto Encyclopedia of Genes and Genomes, look for the relations in this pathway that appear in literature and add weights to these relations accordingly. A higher weight for a relation shows that the relation is much more significant and highly associated for the disease, making it much more desirable to extract and conduct research on.

B. Statement of the Problem

Research has presented that when dealing with complex protein chains of various diseases, there is a large amount of proteins that interact with each other in different ways. When multiple cases of the disease are being studied, some proteins appear in more cases than others. This characteristic makes them much more significant, thus making it more desirable to isolate and study their structure. A particular property worth examining is the connection that exists between the two proteins of interest.

Getting to know the different centralities of a network is also an important prop-

erty of networks to study. The different centralities of the network show significant nodes based on their function.

There are several existing tools that have the capability to show the different centralities of a network and are able to connect these nodes and their interactions, but they do not take into account the number of nodes that participate in the interaction between these two nodes of interest [9]. The interactions that contains these transitive nodes may provide more information to the researcher, or be worth studying itself.

C. Objectives of the Study

The aim of this project is to create a tool that allows a user to upload a graphical representation of a modified protein pathway with weights, compute for its centralities, and be able to generate a subgraph from the loaded graph that contains selected entities and relations. The user should be able to set a threshold for relations and limit for the number of hops that will be considered in the extracted subgraph. The functions that will be available in the tool are the following:

1. The user should be able to:
    (a) Load the file containing the modified KEGG pathway
    (b) See a visual representation of the pathway
    (c) Set a threshold for the relations that will be included in the extracted subgraph
    (d) See the graph that contains all entities connected to the relations that are above the threshold supplied and their direct connections
    (e) Set a limit for the number of edges in between selected entities (hops) and command the program to generate the subgraph
    (f) See the generated subgraph with any indirect relationship between selected entities being represented as a dashed line

(g) Select a indirect relationship and find out the actual path that makes up the relationship

(h) Save the generated subgraph into a .csv file

(i) See the nodes of the loaded pathway colored based on the different centralities of the including:

i. In-degree Centrality

ii. Out-degree Centrality

iii. Betweenness Centrality

iv. Closeness Centrality

v. Eigenvector Centrality

D. Significance of the Project

It is a given that in any biological network, there are various entities whose relationships are of greater interest than others. For a researcher looking for the different ways a network can be affected by external factors, for example a certain drug for Lukemia, being able to isolate and focus on specific proteins that get affected is important. Since biological networks can get complex very easily, it is preferred to use an automated system that can connect and extract these components from network.

The tool presented here provides the researcher a way so that they can find the most significant relations in the network in a systematic way. The subgraphs extracted help in identifying the biomarkers or key components to be considered in the pathways that will be modeled mathematically. Since only a much smaller subnetwork is being used for study, the scope of the research has been narrowed down, maximizing time and minimizing any costs that may come with analyzing a much larger network.

E. Scope and Limitations

1. The user will not be able to modify any part of the graph or its components.
2. The output of subgraph extraction process is not guaranteed to be a connected subgraph all the time.
3. The maximum number of hops that the user may set is only up to three.
4. The input being used comes from a previous process that modifies the KEGG pathway by adding weights to it.
5. The input file is a .csv file with each line being a single relation in the graph.
6. The output file of the subgraph is a .csv file with the same format as the input file.
7. The user will be the one to verify if the provided data in the graph is right or wrong.

F. Assumptions

1. The type of graph that will be used in the problem is a directed graph with costs for each edge.
2. All nodes and edges have non-negative weights.
3. There are no parallel edges in the pathway.

II. Review of Related Literature

One important problem in network analysis is to be able to see if a series of chosen nodes are connected. One possible solution is to check the reachability of the graph in order to see if it is possible to get from one node to another in a graph. Reachability can be computed with the use of algorithm that require only linear time such as breadth first search or iterative deepening search [10]. However, the use of such algorithms is recommended for only a few number of queries.

When finding the reachability matrix of a graph, one is also looking for the transitive closure of the graph.The Transitive Closure Property of graphs is a problem applicable in many fields. Ciampaglia et. al. [11], used the concept of transitive closure for fact checking. A fact is checked by looking if an edge exists between the subject to its object. The authors used the transitive relations of the graph in order to derive the truth value of a statement. Richters and Peixoto [12] studied the properties of trust propagation on networks, based on a simple metric of trust transitivity and found out that the existence of a non-zero fraction of absolute trust is a requirement of the viability of global trust propagation in large systems.

Gene Ontology (GO) provides core biological knowledge representation for modern biologists, whether computationally or experimentally based. The upward propagation of annotations is referred to as computing the transitive closure of the annotation set over the graph of the GO [13, 14, 15]. The use of transitive closure to gather up all annotations relevant to the protein of interest gives the fullest picture of knowledge about the protein and makes best use of the ontological structure.

There are various Transitive Closure algorithms that have been presented in literature. Warshalls algorithm [16] is the most well-known algorithm that can find the transitive closure of a network. Its a simple algorithm that has a running time of O(n3) which makes it inferior to newer algorithms. Munro [17] and Fischer and Meyer [18] presented transitive closure algorithms that use matrix multiplication and that

have the same worst-case execution time as a single matrix multiplication. These are the asymptotically fastest transitive closure algorithms when the input graph is dense (i.e., contains near n2 edges). Nuutila [19] proposed a new algorithm for computing the full transitive closure of a binary relation. The algorithm makes use of the disk memory when the data doesn't fit into the main memory. The algorithm uses a control stack to reduce and localize memory references and uses a single work stack for adjacent components and vertices. Ioannidis et.al. [20] develop two new algorithms (Basic_TC and Gobal_DFTC) which use depth-first search to traverse a graph and a technique called marking to avoid processing some of the arcs in the graph.

Being able to analyze the topological features of a network would give researchers insight on the structure of the network being studied. One topological feature that is commonly used as a basis for finding the importance of a node in a network is the centrality of a node. In biological network, the centrality is used to measure the contribution of a node to the network [5]. Various studies have used the degree and closeness centrality as a means of finding important nodes in the networks [21, 22, 23].

Rio et. al. [24] found out that no single centrality measure identifies essential genes from these networks in a statistically significant way but the combination of at least 2 centrality measures achieves a reliable prediction of most but not all of the essential genes. They have stated no improvement when three or four centrality measures were used.

Another possible use of the centrality measure is to find drug targets. Drug targets are, on average, proteins with high centrality values. According to Mora and Donaldson [25], degree and Centralities seem to be better predictors for cancer. They might not be the best drug target prediction metrics, but after repeated testing, all conclusions drawn were consistent.

Using networks as representation of biological structures is nothing new in the field of systems biology. Before, the dissection of biological networks has occurred

through the efforts of individual laboratories working on one or a few components, limiting a thorough understanding of individual biological processes in the context of the entire cellular network [4]. Various tools have been created by researchers to aid them in the visualization and analysis of these networks.

[26] provides Java graphical interfaces for methodical analysis for both single and comparative transcriptome data without the use of a reference genome (e.g. for non-model organisms). It can import sequences and compute for transitive closure, among other things.

ProViz [27] is a tool for the visualization of protein-protein interaction networks. It provides facilities for navigating in large graphs and exploring biologically relevant features, and adopts emerging standards such as GO and PSI-MI. Its best suited for protein-protein interaction networks and their analysis using different properties. It can also accept plug-ins from the user, providing additional functions based on the users needs.

Cytoscape is an open source software project for integrating biomolecular interaction networks with high-throughput expression data and other molecular states into a unified conceptual framework [28]. It has the capacity to layout and query the network; to visually integrate the network with expression profiles, phenotypes, and other molecular states; and to link the network to databases of functional annotations. The application also accepts plug-ins created by users for more specialized analysis of networks. One such plugin is the NetworkAnalyzer [29]. It computes and displays a comprehensive set of topological parameters.

Another software called PATIKA (Pathway Analysis Tools for Integration and Knowledge Acquisition) [30] provides the user various functions for data integration and pathway analysis. Its designed to provide researchers a solution for modeling and analyzing cellular processes.

BioGateway [31] provides a single entry point to query these resources through

SPARQL. It constitutes a key component for a Semantic Systems Biology approach to generate new hypotheses concerning systems properties.

Pavlopoulos et. al. [9] examine various other network visualization tools and conclude that despite the continuous improvement of visualization algorithms, most existing visualization methods reach limitations in terms of user friendliness when thousands of nodes have to be analyzed and visualized.

III. Theoretical Framework

A. Metabolic Networks

Used for studying and modelling metabolism in various organisms. The metabolic network usually focuses on the mass flow in basic chemical pathways that generate essential components such as amino acids, sugars, and lipids, and the energy required by the biochemical reactions [4].

B. Kyoto Encyclopedia of Genes and Genomes

The Kyoto Encyclopedia of Genes and Genomes (KEGG) is a resource that is used for analysis of gene functions. It is utilized for bioinformatics research and in other related studies. The databases are freely available to the public at http://www.genome.ad.jp/kegg/.

C. Directed Graphs

Directed graphs (or digraphs) consist of a set of nodes V and a set of edges E where all edges which connect a pair of nodes in V are directed. They are often represented in two ways: as a collection of adjacency lists and as an adjacency matrix.

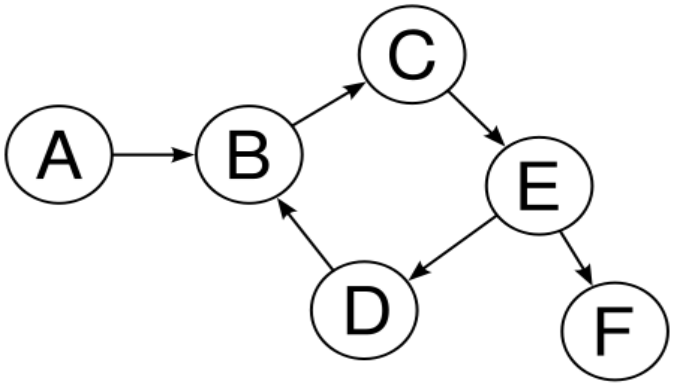

Figure 1: A simple Directed Graph

D. Adjacency Matrix

An adjacency matrix is a finite square matrix that is used to represent a finite graph. The elements in the matrix indicate if a connection between two nodes exist or not. For undirected graphs, the matrix is symmetric.

For a graph composed of the vertex set V, it would have an adjacency matrix of size $|V|$ x $|V|$. The matrix element at i, j would denote if there exists a connection from node i to node j.

E. JGraphT

A free Java graph library that provides mathematical graph theory and algorithms. It has wide support for various kinds of graphs. The library also includes a adapter that could be used with JGraph, now known as JGraphX, a library based on rendering and GUI.The library is available at https://github.com/jgrapht/jgrapht.

F. JGraph

JGraph is a Java Swing library that provide features that are aimed at the display of interactive mathematical graphs. The library is freely available at https://github.com/jgraph/jgraph

G. Matrix Toolkit Java

The Matrix Toolkit Java (MTJ) is an open source Java software library that can perform high performance linear algebra computations. The library is available at https://github.com/fommil/matrix-toolkits-java.

H. Network Centrality

Network Centrality is used as a measure to find out which nodes are important on a network. It is measured by several properties.

1. Degree Centrality

   The node with the most number of edges connected to it is regarded to be important. For graphs that are directed, the centrality may be based on either the in-degree or out-degree characteristic.

2. Betweenness Centrality

   Betweenness is a centrality measure that quantifies the number of times a node acts as a bridge along the shortest path between two other nodes. Nodes that have a high chance of appearing in paths between nodes are considered to be much more central to a network.

   It is given by the expression:

$$b_{st}(v) = \frac{p_{st}(v)}{p_{st}}$$

   where p_{st} is the total number of shortest paths from node s to node t and $p_{st}(v)$ is the number of those paths that pass through v [32].

   Brandes [33] proposes an algorithm for finding the exact betweenness centrality of the vertex in $O(nm + n^2 logn)$.

3. Closeness Centrality

   The closeness of a node is the average length of the shortest path between the node and all other nodes in the network. A node is considered to be more important when it is closer to all the other nodes.

   It was defined by Bavelas[34] as:

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

   Where d(y,x) is the distance between vertices x and y. The result for distances

incoming or outgoing nodes will be different when the graph is directed.

If the graph is disconnected, a common idea is to use the sum of the reciprocal of the distances instead of the reciprocal sum of all the distances [35].

$$C(x) = \sum y \neq x \frac{1}{d(y,x)}$$

4. Eigenvector Centrality

   The Eigenvector shows that the importance of a node is dependent on the importance of its neighbors. It is commonly regarded as a ranking measure.

   The eigenvector can be given by:

$$x_i = \frac{1}{\lambda} \sum_k a_{k,i} x_k$$

   Where x_i is the eigenvector centrality of node i, A = $(a_{i,j})$ is the adjacency matrix of a graph and λ is a constant not equal to zero.

IV. Design and Implementation

A. Use Cases

The user should be able to view the Pathway as a graph and generate a subgraph from it that connects nodes and edges that surpass a threshold. The subgraph also has a limited number of transitive nodes in between the required nodes that is given by the user. The user can get the different topological properties of the generated subgraph.

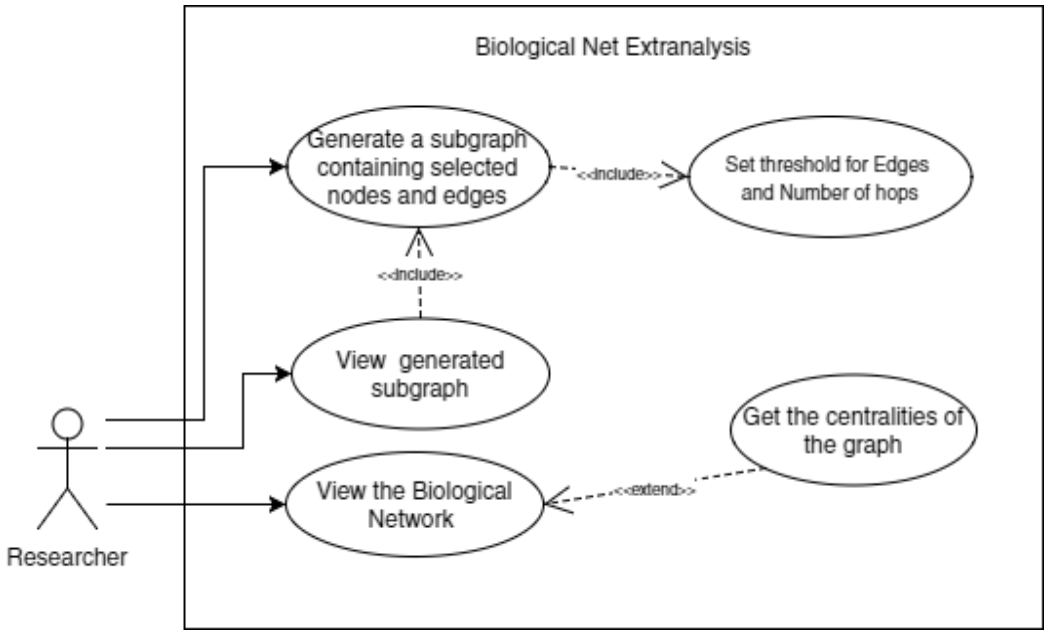

Figure 2: The Use-case Diagram of the Biological Net Extranalysis

B. Flow Chart

The user selects the file containing the graph to be loaded into the program. A visual representation of the graph is then drawn onto the screen. The user may choose to input a threshold for the edges to be included in the subnetwork. The user is then shown the forest containing nodes that exceed the set threshold. The user will now input the number of hops for the subgraph. The subgraph generated now contains

the edges that exceed the threshold and hops that are within the limit the user put. A visual representation of the subgraph is presented on the screen and the user is given the choice to export the subgraph. An additional function of the tool is to get the centrality measures of the graph which will color the nodes in the graph based on the computed centrality value.

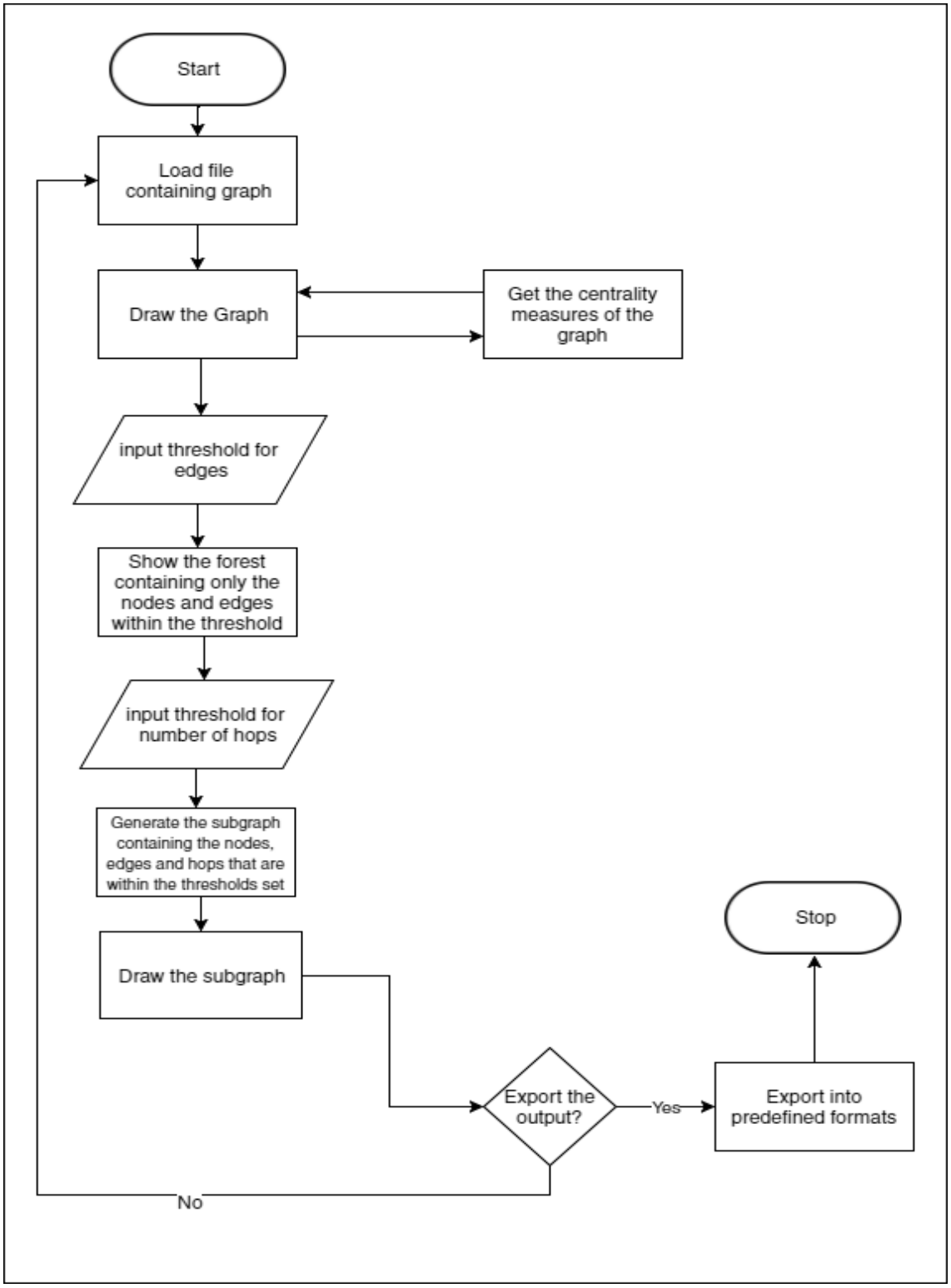

Figure 3: Flowchart Diagram of the Biological Net Extranalysis

C. System Architecture

The tool was written using Java. The following libraries were used:

1. **JGraphT**

   This free Java graph library is used primarily to store the graph object and perform calculations on it. It also has a number of graph algorithms included.

2. **JGraph**

   Another free Java graph library with the focus given on the rendering of the graph for display. This is the library used to show the visual representation of the graph to the user.

3. **Matrix Toolkit Java (MTJ)**

   An open source Java library software that can perform linear algrebra computations. It is primarily used in this project to compute for the Eigenvector centrality of the nodes in the graph.

D. Technical Architecture

The tool is a stand-alone desktop application that is run on Java. The tool only requires the local memory of the machine.

Minimum System Requirements

1. A midrange CPU with at least 1.6 GHz
2. At least 2.0 GB RAM
3. A 64-bit Operating System
4. Windows 7/8/10
5. Java Runtime Environment

V. Results

The Biological Net Extranalysis is a tool that visualizes Biological Networks, performs analysis on the graph and allows the user to extract a subgraph containing the essential nodes or relations in the graph.

When the tool is loaded, the splash screen is initially shown then the Main Window of the program is presented to the user.

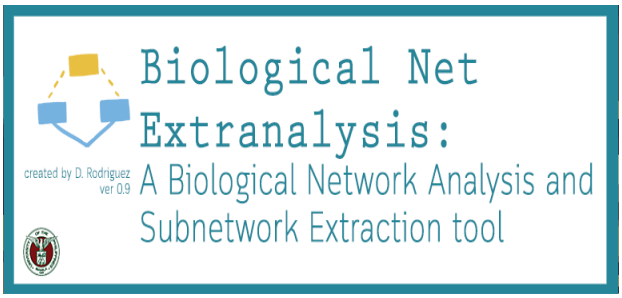

Figure 4: Splash Screen when the program starts

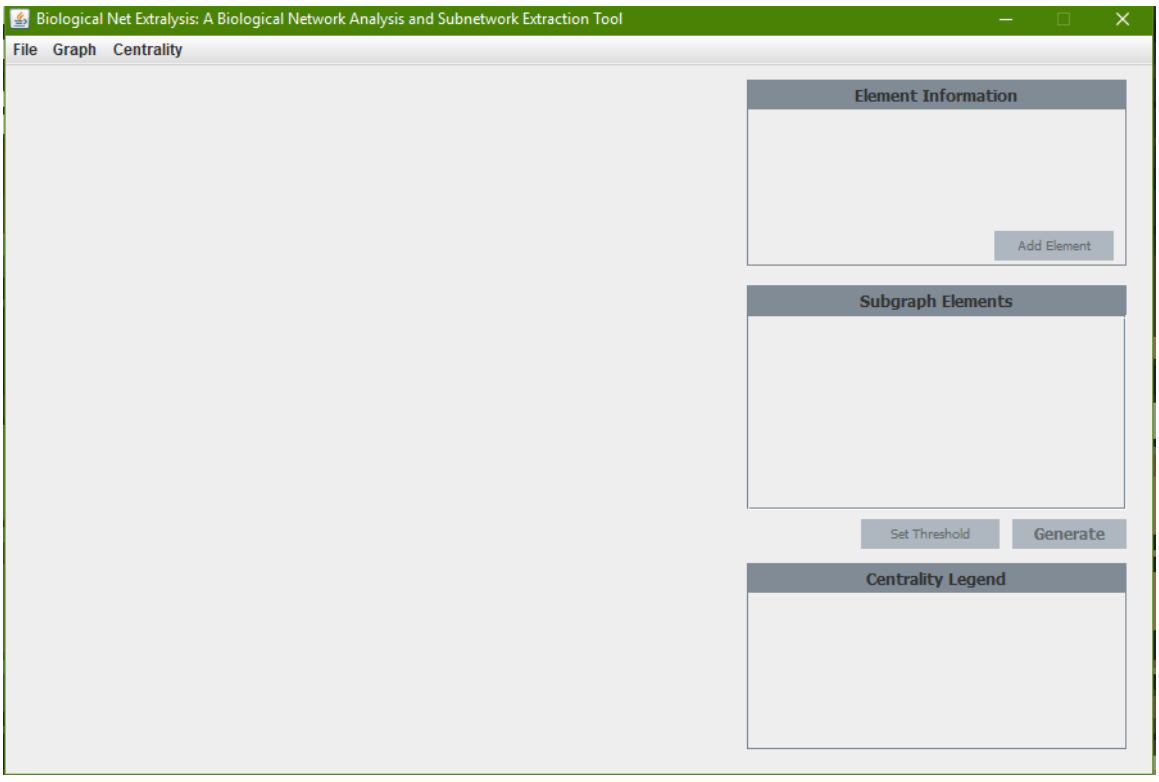

Figure 5: Main Program Window of the Biological Net Extranalysis

A. Loading the File

The user selects the "Import Graph..." option in the "File" Menu and a window pops up for the File Selection.

Once the file is selected, the graph is now loaded into the program and the visual representation of the graph is now shown to the user.

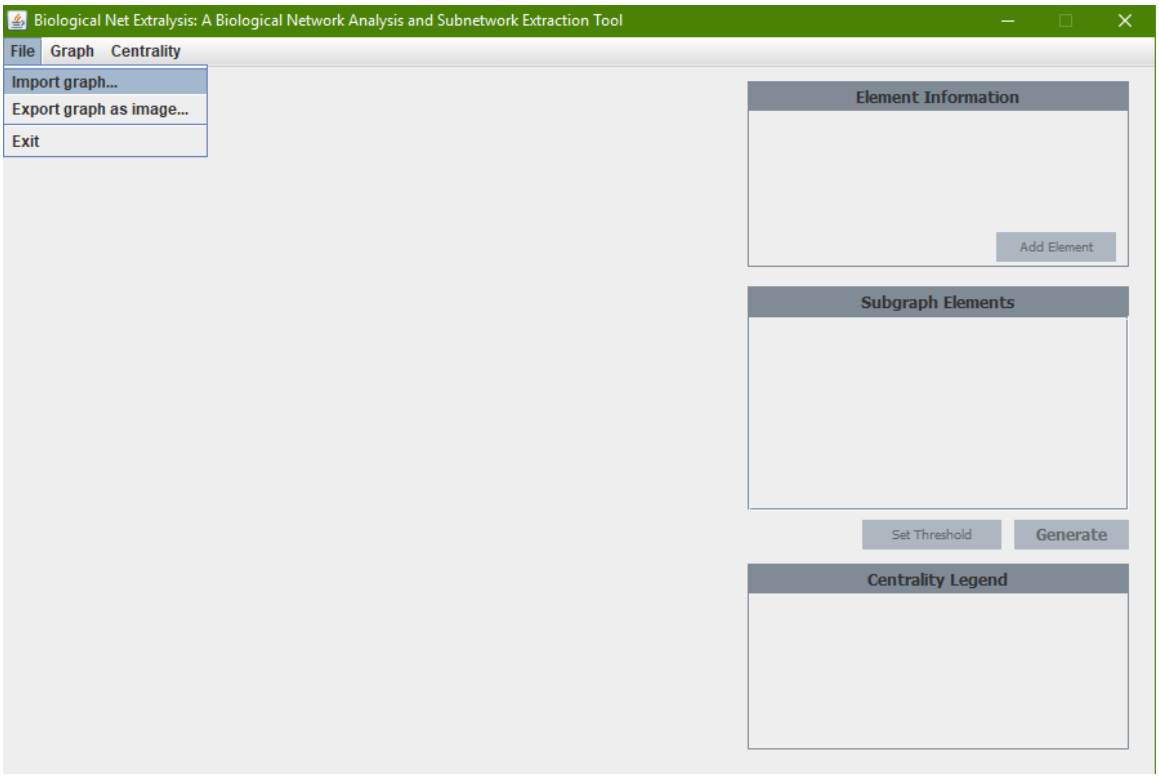

Figure 6: The File Menu containing the option to load the graph

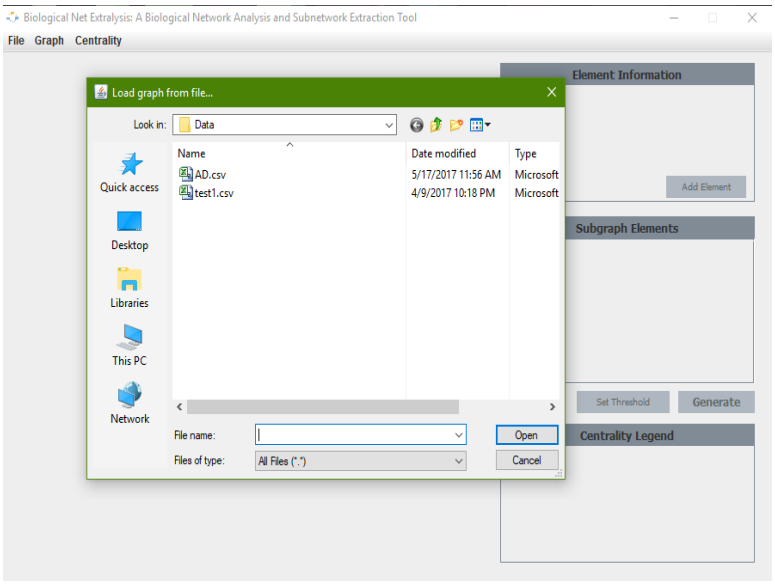

Figure 7: The window that pops up for file selection

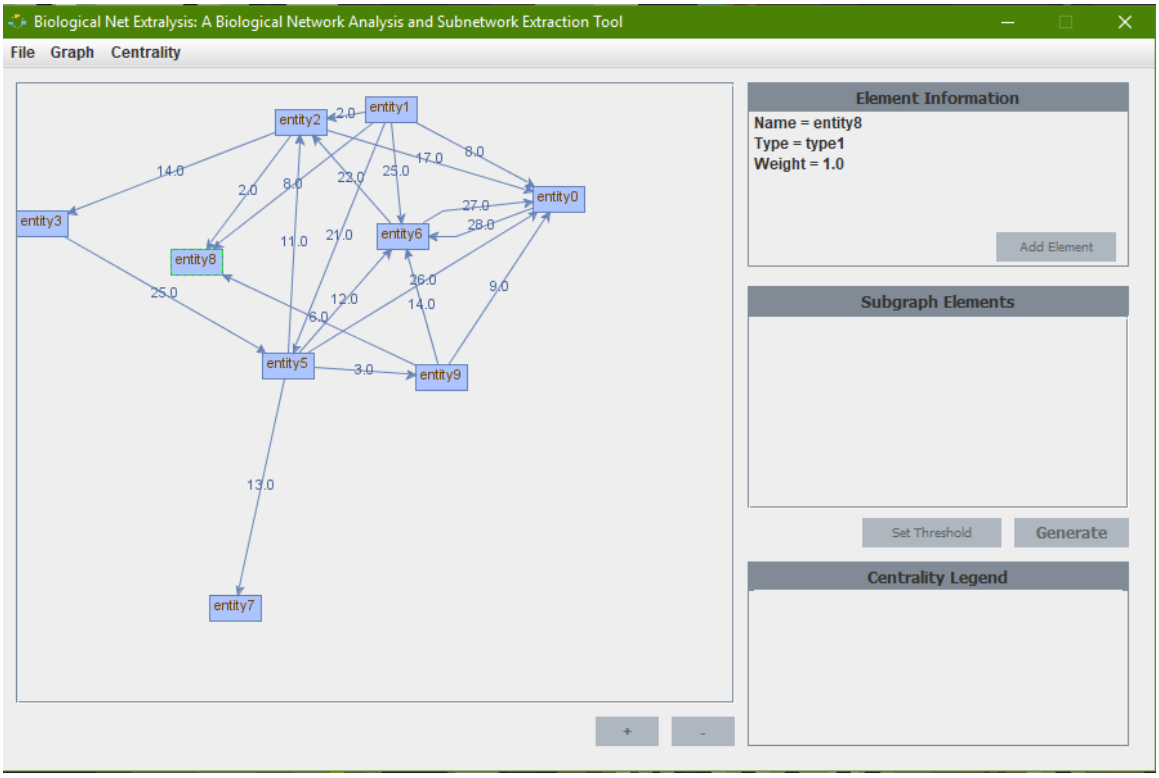

Figure 8: The loaded graph with Entity8 highlighted

B. Setting the Threshold for the Edges

The user can now set the threshold of edges that will be included in the graph. If an edge goes above the set threshold, the source and target node of the edge are included among the selected nodes of the subgraph.

Selected Nodes are now highlighted in the original graph to make it easier to spot them.

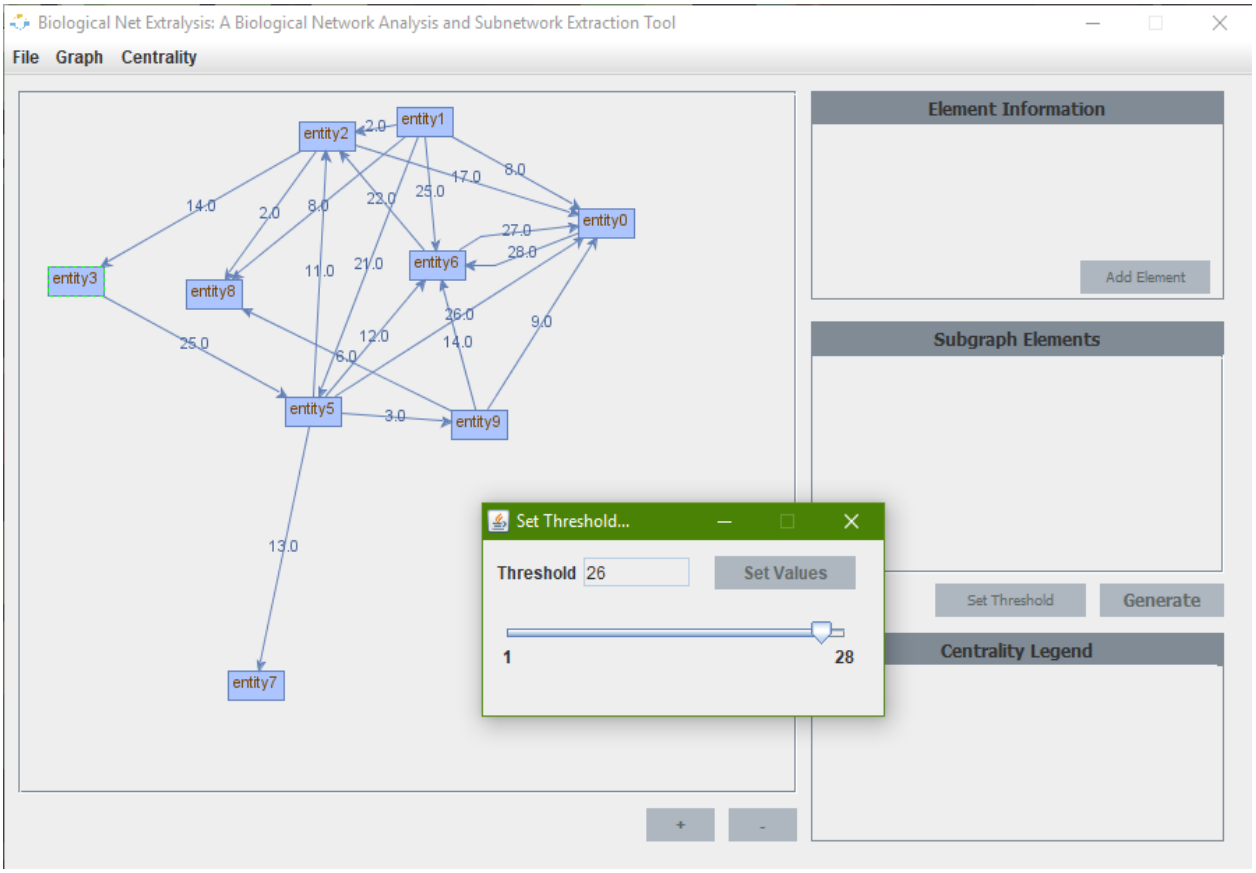

Figure 9: The user setting the threshold of edge weight

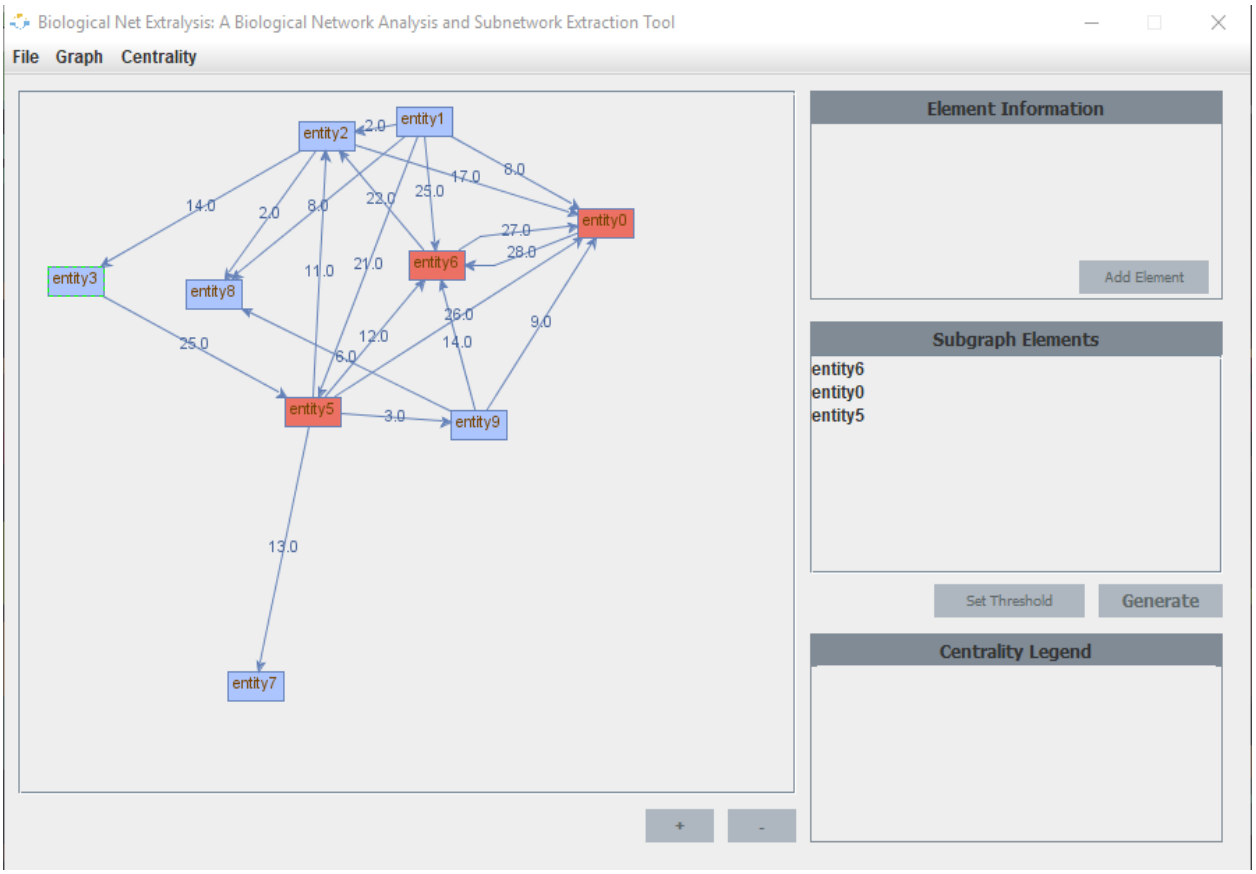

Figure 10: The nodes in edges that exceed the threshold are now highlighted and colored differently

C. Setting the Limit for the Hops

The user is now shown the subgraph with only the direct edges between nodes present. They now set the number of hops that act as the limit of the number of edges in between selected nodes of the graph.

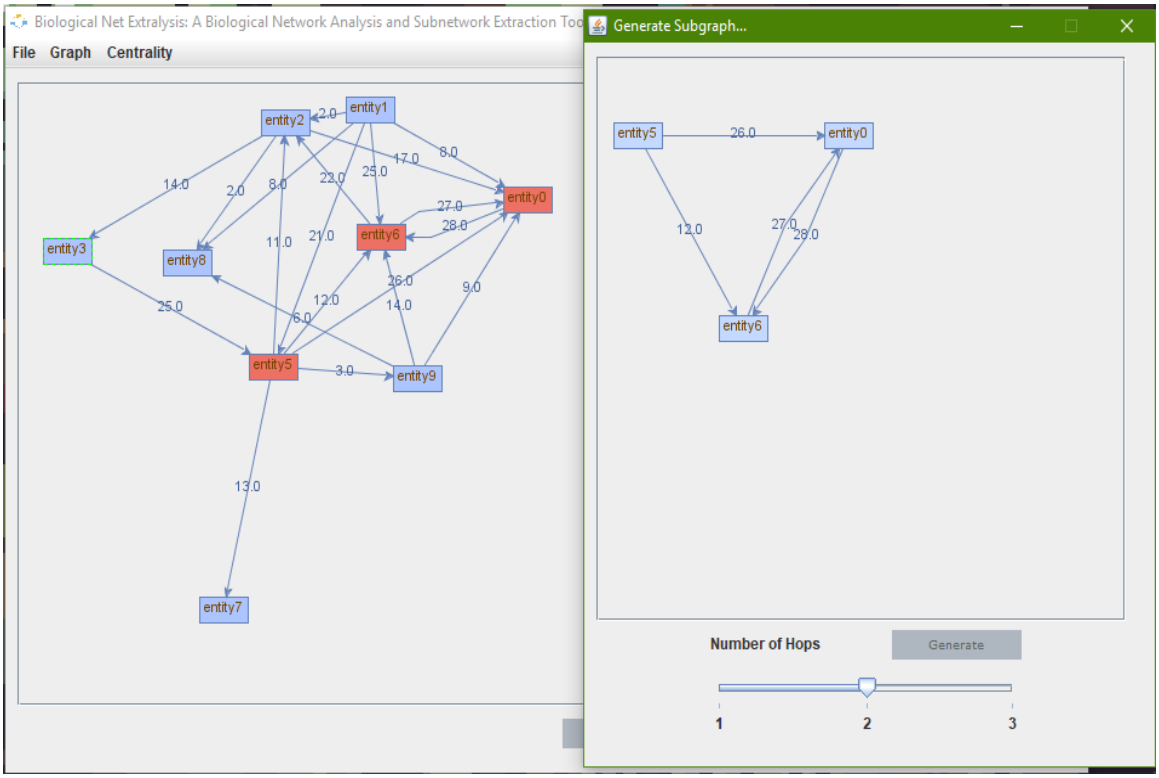

Figure 11: The window containing the initial graph/forest and where the user can set the number of hops to be included in the subgraph

D. Generating and Showing the Subgraph

With the input threshold for the Edge Weight and the Hops, a subgraph is now generated showing the selected nodes and any indirect relationship being showed as a dashed line.

The user may click on the edge in order to expand the edge and examine the components that make up the edge.

There is also the option to export the Subgraph into a .csv file that can be read by the tool for further analysis or subgraph extraction.

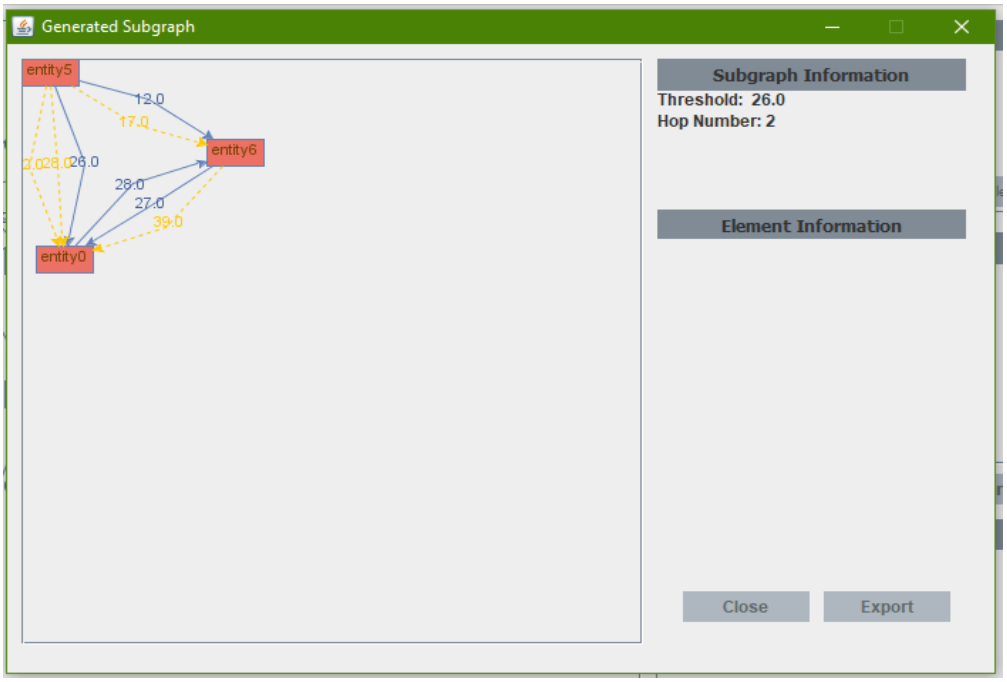

Figure 12: The generated subgraph with any indirect edge being represented as a dashed line

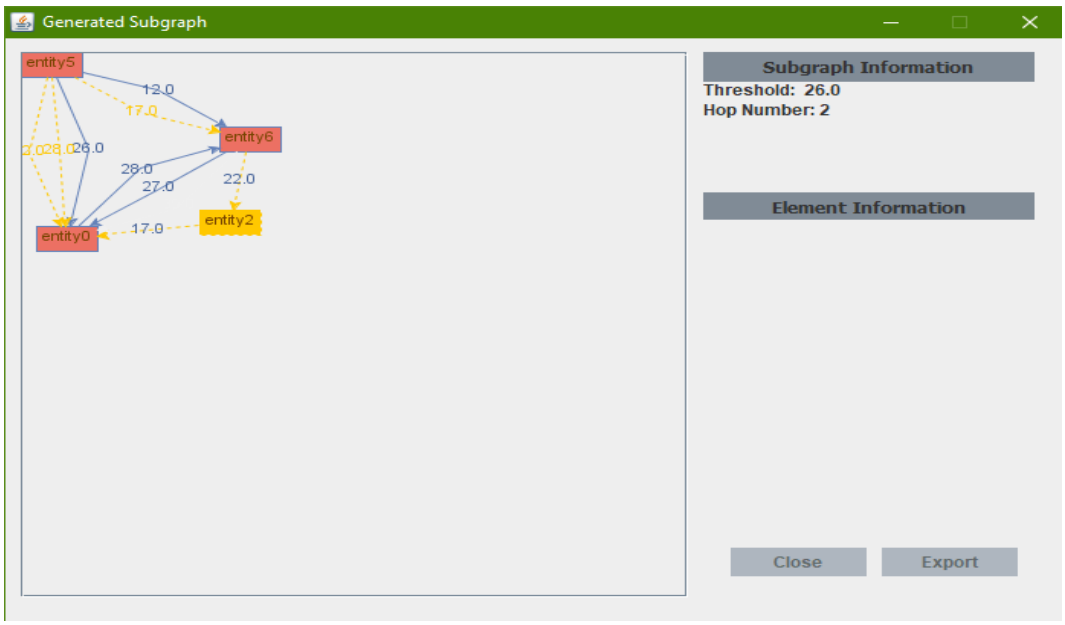

Figure 13: An indirect edge that has been "unfolded" by the user, showing the composition of the edge

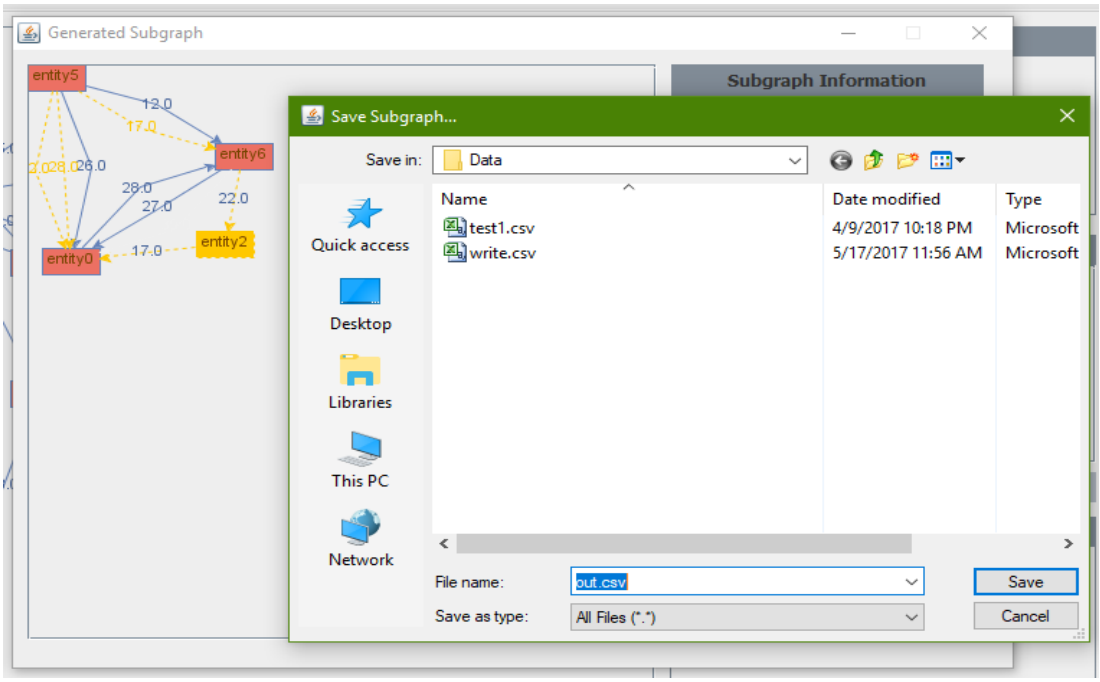

Figure 14: The window that pops up for the exporting the graph

E. Adding Specific Elements

The user may also add specific nodes or edges that they wish to include in the subgraph by selecting on the node or edge and clicking on the "Add Element" button.

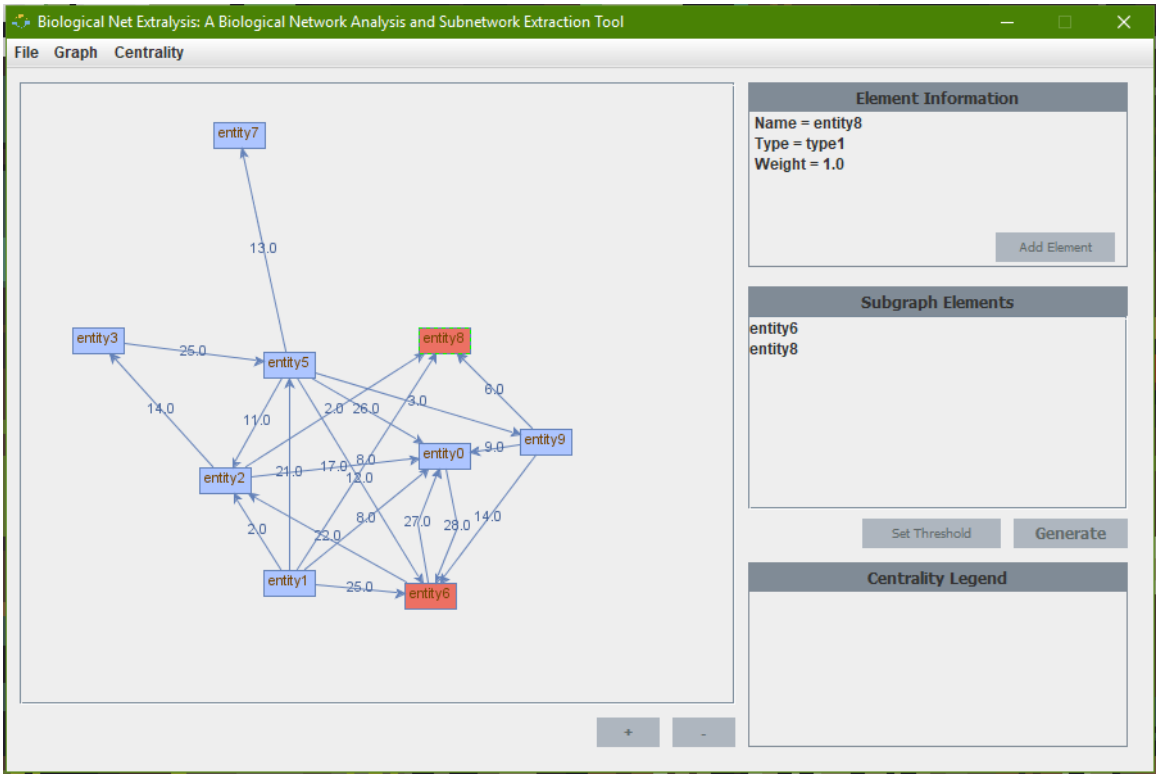

Figure 15: The user may also add elements they wish to add with the "Add Element Button"

F. Computing Centrality Measures

With the main graph loaded into the tool, the user may compute for the different centrality measures that the graph has. This colors the different nodes of the graph based on the computed value.

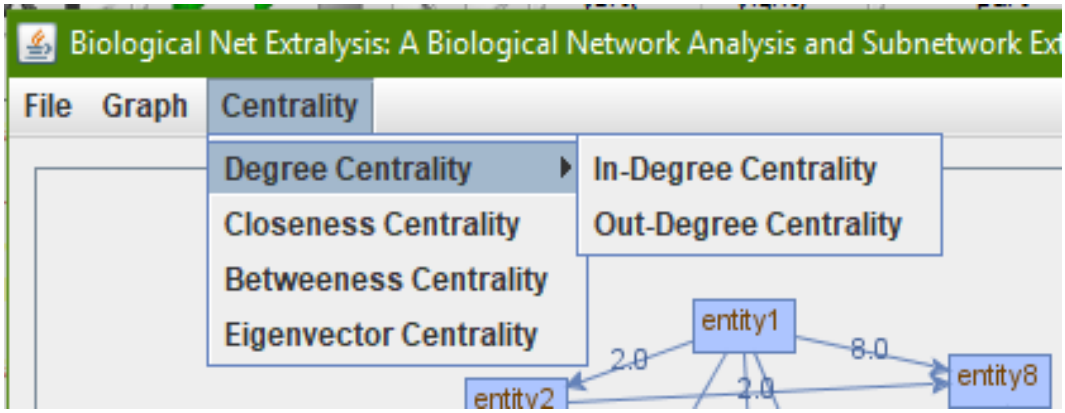

Figure 16: The different centralities that can be computed in the graph

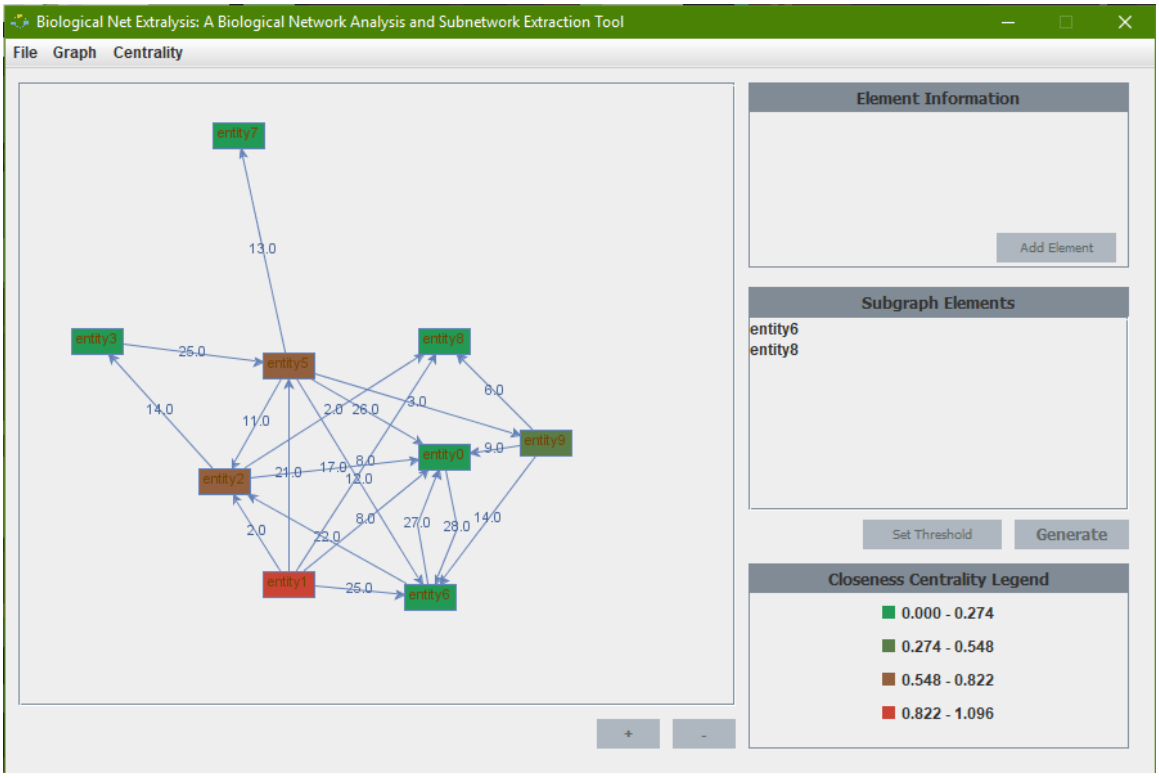

Figure 17: The graph's nodes colored based on the Betweenness Centrality metric

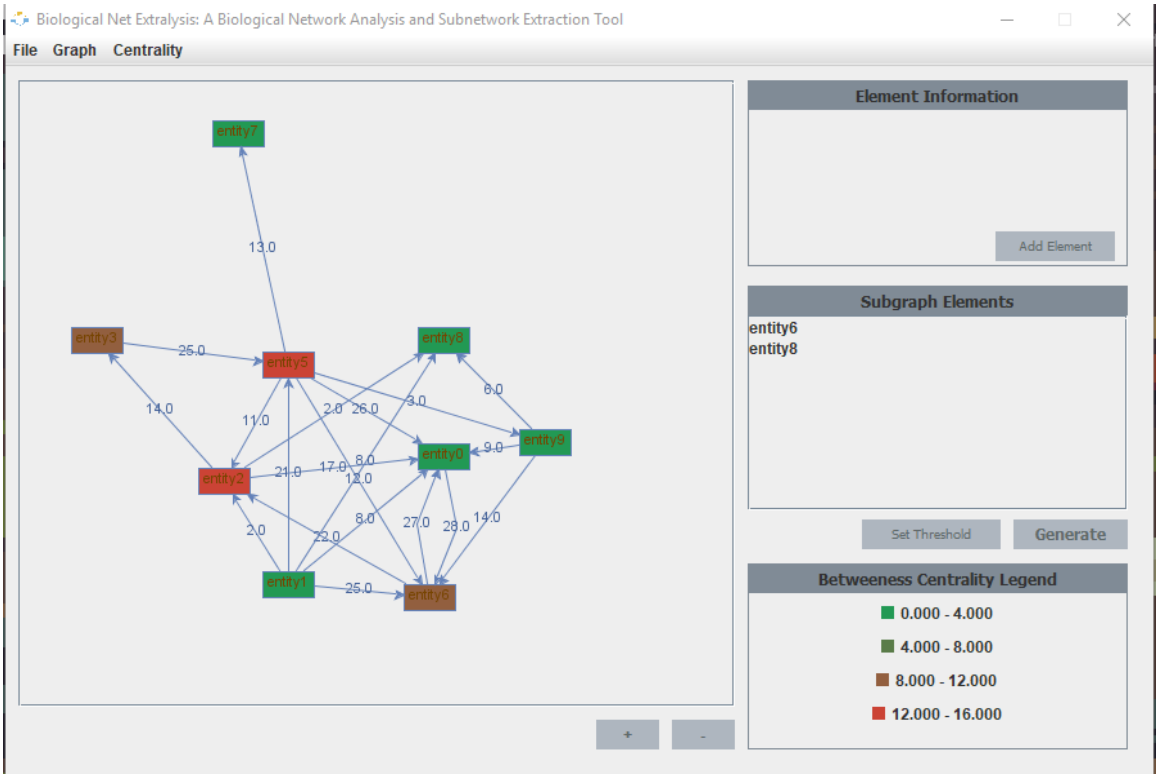

Figure 18: The graph's nodes colored based on the Closeness Centrality metric

G. Exporting Graph as Image

Under the File menu, there is an option wherein the user may export the visual representation of the graph as a .png file.

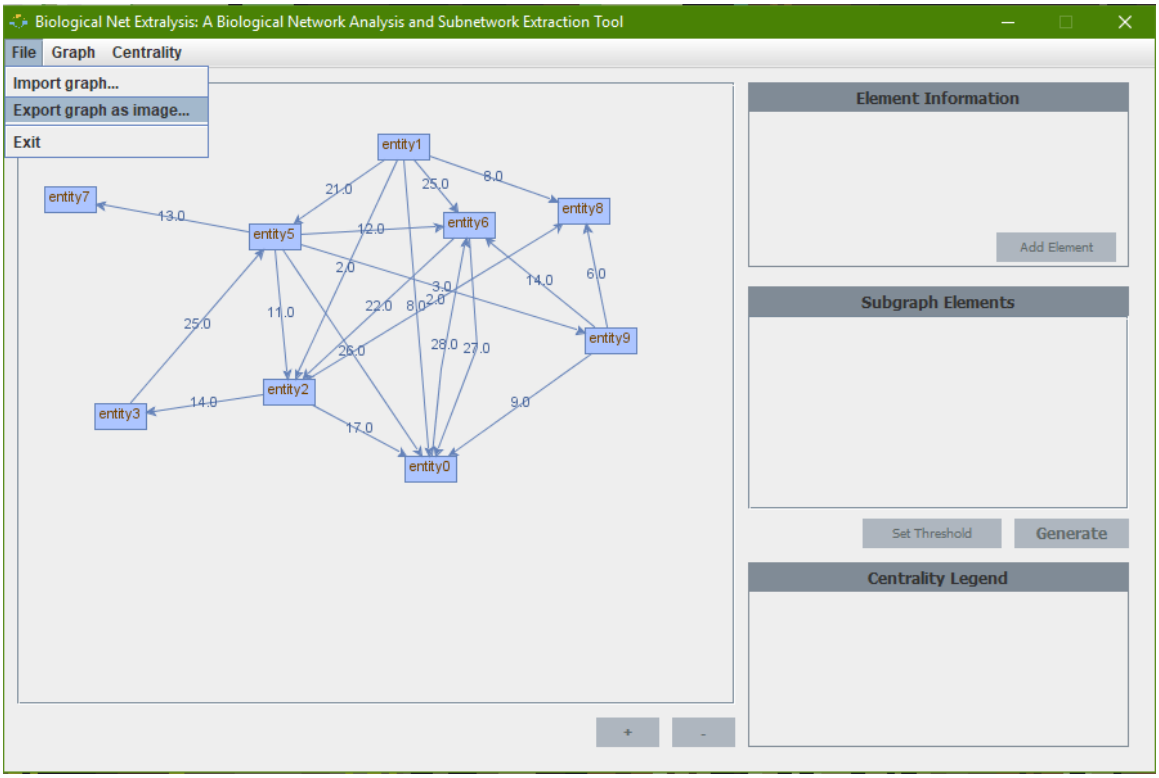

Figure 19: The user may also export the representation of the graph as a png file

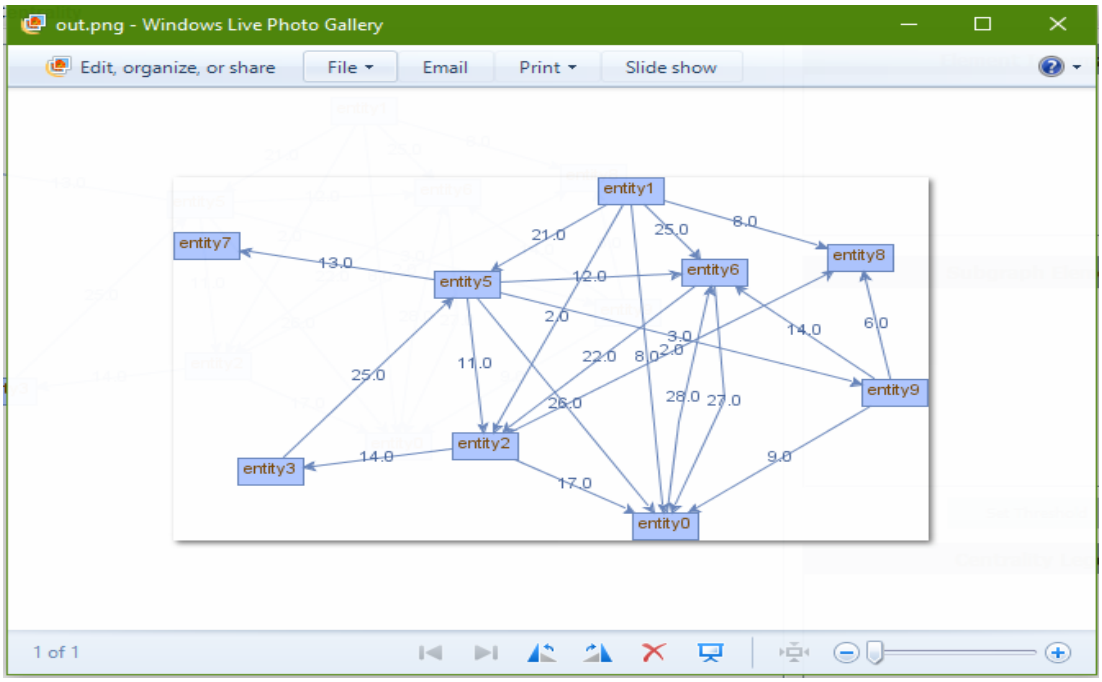

Figure 20: The exported image of the graph

VI. Discussions

The Biological Net Extranalysis is a tool that helps the user visualize a graph, extract a subnetwork containing the relationships that are significant in the network, and perform network analysis on the graph.

The user loads the graph into the program and is presented with a visual representation of the graph. A Java library, jGrapht, is being used to contain the graph object in the program and provide different algorithms that will be used for various computations. The library jGraph is used to create the visual representation that is presented to the user.

The loaded graph can have two things done on it: have a subgraph of selected nodes extracted or show the graph with its nodes colored based on the computed centrality values.

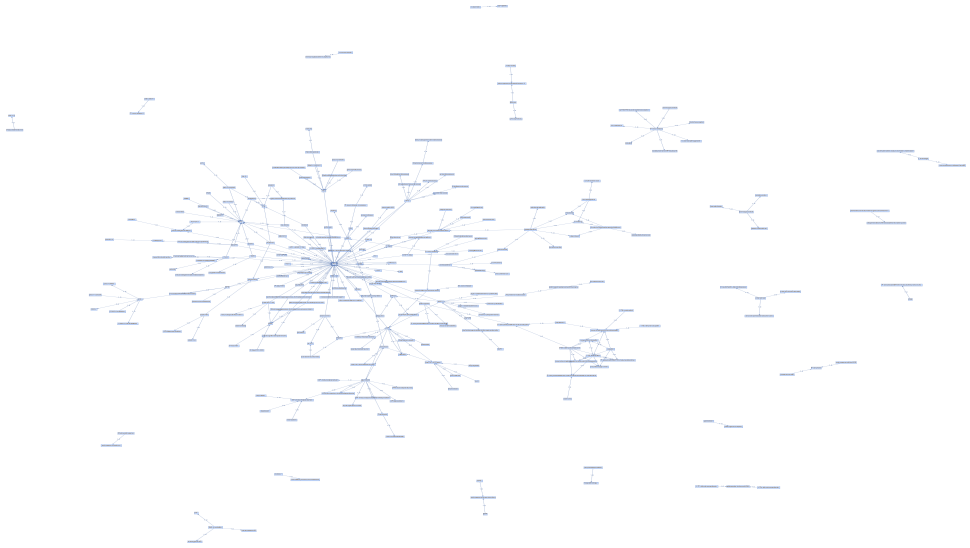

Figure 21: The image of the graph containing the relations found in Alzheimer's Disease

The tool was used to run a test on the incomplete modified KEGG relations of the Alzheimer's Disease that contains various weights. The weights in the graph represent the number of times a certain edge is cited in literature. The weights in

this test graph are based on 12 manually annotated and validated articles. The data hasn't been normalized. Using the tool to extract edges with the heaviest weights, it can be shown that there is an indirect relation between glycitein and Na+/K+ATPase with amyloid being a node in this relation.

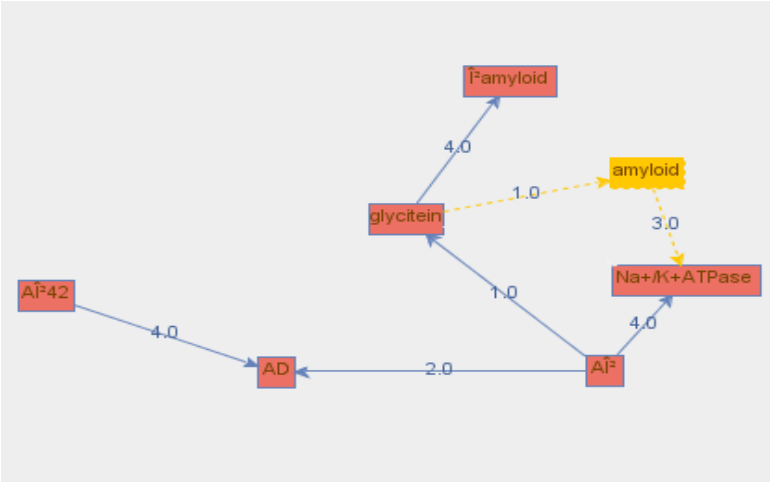

Figure 22: The subgraph generated from getting the heaviest weighted edge from the graph

Given this result, the relationship between amyloid and glycitein may be a subject of interest since the low weight that the edge has shows that there isn't too much reference regarding this node, thus making a point of interest for further research. Additionally, the amyloid and Na+/K+ATPase may also be worth looking into since the heavier weight shows that the relationship is studied more and therefore may have an significant part in the network.

When the different centralities of the network are performed on the graph, it is consistently shown that the central node, AD (Alzheimer's Disease), is colored with the highest values. There are other nodes that are also colored such that they have relatively higher centrality values than the other nodes and may be worth looking into.

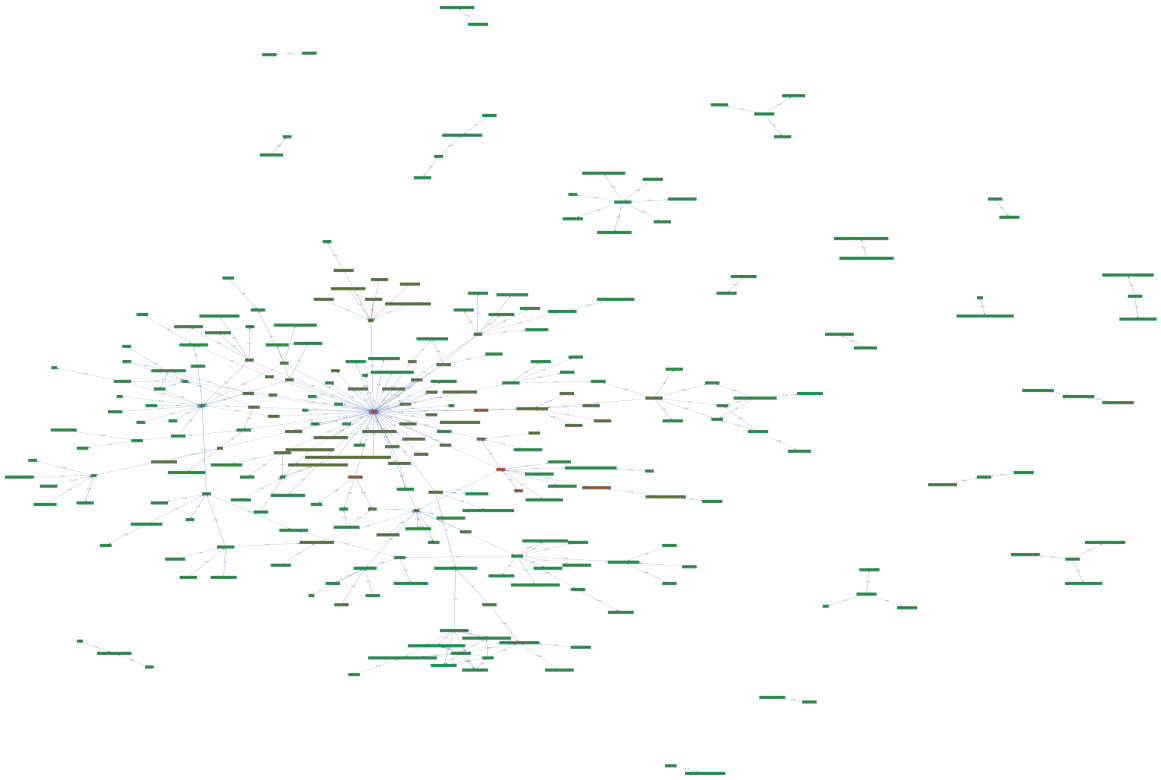

Figure 23: The graph's nodes colored based on the Eigenvector centrality values

VII. Conclusions

The tool created, Biological Net Extranalysis, lets the user load a graph from a file and shows a visual representation of the graph to the user. The user is able to create a subnetwork from selected edges that surpass a certain threshold, and examine any indirect relations these nodes have. The program makes use of the concept of transitivity in order to find the different indirect relationships that the nodes may have. The user may export the generated subgraph for further analysis and study.

The tool also computes for the different centrality measures of the graph, including (i) In-degree, (ii) Out-degree, (iii) Betweenness, (iv) Closeness, and (v) Eigenvector. It colors the nodes of the graph according to the calculated value. Being able to find these centralities may help the user find nodes that are important to a network depending on the mechanism present.

In conclusion, this tool is a handy way to be able to extract nodes of interest from a network and examine the indirect relationships they may have with each other. With researchers looking for ways to minimize the scope of things to experiment on, being able to extract the subgraph is certainly one way to minimize the size of what to should be studied.

VIII. Recommendations

The tool can be improved by increasing the limit for the number of hops that the user can set. Looking for more efficient algorithms to extract the subgraph will certainly aid in cutting the increased computation time that comes when increasing the limit for the number of hops. There are various transitive algorithms for graphs proposed that may be used.

The tool can also benefit from generating a subgraph based on the centrality measures that were computed. Extracting certain nodes based on their centrality values may show interesting pathways and relationships that are worth looking into.

Another thing that can be improved is looking for a way to present the graph in a much more visually pleasing way. As of the current version, the nodes and edges are randomly placed, making things look a bit messy. It might be worth looking up planar graph theory for this.

As of now, the tool can only load a specific graph format that it was originally created for, which are weighted KEGG pathways. Increasing its capabilities such that it can read other Biological networks, like microarrays, could make the tool useful in more applications in the field.

IX. Bibliography

[1] T. Aittokallio and B. Schwikowski, "Graph-based methods for analysing networks in cell biology," *Briefings in bioinformatics*, vol. 7, no. 3, pp. 243–255, 2006.

[2] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Müller, "Identifying functional modules in protein–protein interaction networks: an integrated exact approach," *Bioinformatics*, vol. 24, no. 13, pp. i223–i231, 2008.

[3] T. Aittokallio and B. Schwikowski, "Graph-based methods for analysing networks in cell biology," *Briefings in bioinformatics*, vol. 7, no. 3, pp. 243–255, 2006.

[4] X. Zhu, M. Gerstein, and M. Snyder, "Getting connected: analysis and principles of biological networks," *Genes & development*, vol. 21, no. 9, pp. 1010–1024, 2007.

[5] G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos, "Using graph theory to analyze biological networks," *BioData mining*, vol. 4, no. 1, p. 1, 2011.

[6] L. Yang, K. Tang, Y. Qi, H. Ye, W. Chen, Y. Zhang, and Z. Cao, "Potential metabolic mechanism of girls' central precocious puberty: a network analysis on urine metabonomics data," *BMC systems biology*, vol. 6, no. 3, p. 1, 2012.

[7] J. Zhao, C. Geng, L. Tao, D. Zhang, Y. Jiang, K. Tang, R. Zhu, H. Yu, W. Zhang, F. He, *et al.*, "Reconstruction and analysis of human liver-specific metabolic network based on cnhlpp data," *Journal of proteome research*, vol. 9, no. 4, pp. 1648–1658, 2010.

[8] A. Mazza, I. Gat-Viks, H. Farhan, and R. Sharan, "A minimum-labeling approach for reconstructing protein networks across multiple conditions," *Algorithms for Molecular Biology*, vol. 9, no. 1, p. 1, 2014.

[9] G. A. Pavlopoulos, A.-L. Wegener, and R. Schneider, "A survey of visualization tools for biological network analysis," *Biodata mining*, vol. 1, no. 1, p. 1, 2008.

[10] J. L. Gersting, *Mathematical structures for computer science*. Macmillan, 2007.

[11] G. L. Ciampaglia, P. Shiralkar, L. M. Rocha, J. Bollen, F. Menczer, and A. Flammini, "Computational fact checking from knowledge networks," *PloS one*, vol. 10, no. 6, p. e0128193, 2015.

[12] O. Richters and T. P. Peixoto, "Trust transitivity in social networks," *PloS one*, vol. 6, no. 4, p. e18384, 2011.

[13] J. A. Blake, "Ten quick tips for using the gene ontology," *PLoS Comput Biol*, vol. 9, no. 11, p. e1003343, 2013.

[14] N. H. Shah, T. Cole, and M. A. Musen, "Analyses using disease ontologies," *PLoS Comput Biol*, vol. 8, no. 12, p. e1002827, 2012.

[15] A. Oellrich, N. Collier, D. Smedley, and T. Groza, "Generation of silver standard concept annotations from biomedical texts with special relevance to phenotypes," *PloS one*, vol. 10, no. 1, p. e0116040, 2015.

[16] S. Warshall, "A theorem on boolean matrices," *Journal of the ACM (JACM)*, vol. 9, no. 1, pp. 11–12, 1962.

[17] I. Munro, "Efficient determination of the transitive closure of a directed graph," *Information Processing Letters*, vol. 1, no. 2, pp. 56–58, 1971.

[18] M. J. Fischer and A. R. Meyer, "Boolean matrix multiplication and transitive closure," in *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pp. 129–131, IEEE, 1971.

[19] V. Hirvisalo, E. Nuutila, and E. Soisalon-Soininen, "Transitive closure algorithm disk tc and its performance analysis," 1996.

[20] Y. Ioannidis, R. Ramakrishnan, and L. Winger, "Transitive closure algorithms based on graph traversal," *ACM Transactions on Database Systems (TODS)*, vol. 18, no. 3, pp. 512–576, 1993.

[21] T.-P. Nguyen, M. Scotti, M. J. Morine, and C. Priami, "Model-based clustering reveals vitamin d dependent multi-centrality hubs in a network of vitamin-related proteins," *BMC systems biology*, vol. 5, no. 1, p. 1, 2011.

[22] K. Ning, H. K. Ng, S. Srihari, H. W. Leong, and A. I. Nesvizhskii, "Examination of the relationship between essential genes in ppi network and hub proteins in reverse nearest neighbor topology," *BMC bioinformatics*, vol. 11, no. 1, p. 1, 2010.

[23] M. Sajitz-Hermstein and Z. Nikoloski, "Functional centrality as a predictor of shifts in metabolic flux states," *BMC research notes*, vol. 9, no. 1, p. 317, 2016.

[24] G. del Rio, D. Koschützki, and G. Coello, "How to identify essential genes from molecular networks?," *BMC Systems Biology*, vol. 3, no. 1, p. 1, 2009.

[25] A. Mora and I. M. Donaldson, "Effects of protein interaction data integration, representation and reliability on the use of network properties for drug target prediction," *BMC bioinformatics*, vol. 13, no. 1, p. 1, 2012.

[26] C. Soderlund, W. Nelson, M. Willer, and D. R. Gang, "Tcw: Transcriptome computational workbench," *PLoS One*, vol. 8, no. 7, p. e69401, 2013.

[27] F. Iragne, M. Nikolski, B. Mathieu, D. Auber, and D. Sherman, "Proviz: protein interaction visualization and exploration," *Bioinformatics*, vol. 21, no. 2, pp. 272–274, 2005.

[28] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker, "Cytoscape: a software environment for inte-

grated models of biomolecular interaction networks," *Genome research*, vol. 13, no. 11, pp. 2498–2504, 2003.

[29] Y. Assenov, F. Ramírez, S.-E. Schelhorn, T. Lengauer, and M. Albrecht, "Computing topological parameters of biological networks," *Bioinformatics*, vol. 24, no. 2, pp. 282–284, 2008.

[30] E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, G. Nisanci, R. Cetin-Atalay, and M. Ozturk, "Patika: an integrated visual environment for collaborative construction and analysis of cellular pathways," *Bioinformatics*, vol. 18, no. 7, pp. 996–1003, 2002.

[31] E. Antezana, W. Blondé, M. Egaña, A. Rutherford, R. Stevens, B. De Baets, V. Mironov, and M. Kuiper, "Biogateway: a semantic systems biology tool for the life sciences," *BMC bioinformatics*, vol. 10, no. 10, p. 1, 2009.

[32] D. R. White and S. P. Borgatti, "Betweenness centrality measures for directed graphs," *Social Networks*, vol. 16, no. 4, pp. 335–346, 1994.

[33] U. Brandes and D. Fleischer, "Centrality measures based on current flow," in *Annual Symposium on Theoretical Aspects of Computer Science*, pp. 533–544, Springer, 2005.

[34] A. Bavelas, "Communication patterns in task-oriented groups.," *Journal of the acoustical society of America*, 1950.

[35] A. Dekker, "Conceptual distance in social network analysis," *Journal of Social Structure*, vol. 6, no. 3, 2005.

X. Appendix

A. Source Code

WeightedVertex.java

```
package graph;

import java.io.Serializable;

/**
* The WeightedVertex class which stands as the representation of one entity in the graph.
* @author Danielle Rodriguez
*
*/
@SuppressWarnings("serial")
public class WeightedVertex implements Comparable<WeightedVertex>, Serializable {
private String name;
private String type;
private double weight;

/**
* Constructor for WeightedVertex. Type is left empty.
*
* @param name - Name of the Vertex
* @param weight - Weight of the Vertex
*/
public WeightedVertex(String name, double weight){
initialize(name, "", weight);
}

/**
* Constructor for WeightedVertex. Weight is automatically set to 1.
*
* @param name - Name of the Vertex
* @param type - Type of the Vertex
*/
public WeightedVertex(String name, String type){
initialize(name, type, 1.0);
}

/**
* Constructor for WeightedVertex.
* @param name - Name of the Vertex
* @param type - Type of the Vertex
* @param weight - Weight of the Vertex
*/
public WeightedVertex(String name, String type, double weight){
initialize(name, type, weight);
}

/**
* Set the name of the Vertex.
* @param name - Name of the Vertex
*/
public void setName(String name){
this.name = name;
}

/**
* Set the type of the Vertex.
* @param type - Type of the Vertex
*/
public void setType(String type){
this.type = type;
}

/**
* Set the Weight of the Vertex.
* @param weight - Weight of the Vertex
*/
public void setWeight(double weight){
this.weight = weight;
}

/**
* Get the Name of the Vertex.
* @return String containing the Name of the Vertex
*/
public String getName(){
return name;
}

/**
* Get the Type of the Vertex.
* @return String containing the Type of the Vertex
```

```
*/
public String getType(){
return type;
}

/**
* Get the Weight of the Vertex.
* @return double containing the weight of the vertex
*/
public double getWeight(){
return weight;
}

@Override
public int hashCode(){
return name.length();
}

@Override
public String toString(){
return name;
}

@Override
public boolean equals(Object other){
if(other == null) return false;
if(other == this) return true;
if(!(other instanceof WeightedVertex)) return false;
WeightedVertex otherVertex = (WeightedVertex) other;
if(name.equals(otherVertex.getName())) return true;
else return false;

}

@Override
public int compareTo(WeightedVertex o) {
return name.compareTo(o.getName());
}

public int compare(WeightedVertex v){
if(v.getWeight() < weight)
return 1;
else if(v.getWeight() == weight)
return 0;
else
return -1;
}

private void initialize(String name, String type, double weight){
this.name = name;
this.type = type;
this.weight = weight;
}

}
```

WeightedEdgeWithType.java

```
package graph;

import org.jgrapht.graph.DefaultWeightedEdge;

/**
* The Edge that is used in the graph. It is a Weighted Edge with a type added to it.
*
* @author Danielle Rodriguez
*
*/
@SuppressWarnings("serial")
public class WeightedEdgeWithType extends DefaultWeightedEdge {
private String type;

/**'
* Set the type of the edge.
*
* @param type - String that is the type of the edge.
*/
public void setType(String type){
this.type = type;
}

/**
*  Get the type of the edge.
*
* @return String that is the type of the edge.
*/
public String getType(){
return type;
}

@Override
public String toString(){
```

```
return Double.toString(super.getWeight());
}

}
```

VertexPicker.java

```
package extractor;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import org.jgrapht.graph.DirectedWeightedMultigraph;
import org.jgrapht.traverse.DepthFirstIterator;
/**
* Class for choosing the Vertices that go over the set thresholds
*
* @author Danielle Rodriguez
*
*/
public class VertexPicker {
private DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;

public VertexPicker(DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph){
this.graph = graph;
}

public Set<WeightedVertex> getChosenVertices(double vertexWeight, double edgeWeight){
Set<WeightedVertex> listOfVertices = getVerticesOverWeightBoundary(vertexWeight);

List<WeightedEdgeWithType> listOfEdges = getEdgesOverWeightBoundary(edgeWeight);
Set<WeightedVertex> vertexFromEdges = getVertexSetFromEdges(listOfEdges);

listOfVertices.addAll(vertexFromEdges);

return listOfVertices;
}

public Set<WeightedVertex> getChosenVertices(double weight){
return getChosenVertices(weight, weight);
}

private Set<WeightedVertex> getVerticesOverWeightBoundary(double weightBoundary){
Set<WeightedVertex> verticesOver = new HashSet<WeightedVertex>();
Iterator<WeightedVertex> vertexIter = new DepthFirstIterator<>(graph);
while(vertexIter.hasNext()){
WeightedVertex curVertex = vertexIter.next();
if(curVertex.getWeight() >= weightBoundary)
verticesOver.add(curVertex);
}
return verticesOver;
}

private List<WeightedEdgeWithType> getEdgesOverWeightBoundary(double weightBoundary){
List<WeightedEdgeWithType> edgesOver = new ArrayList<WeightedEdgeWithType>();
for(WeightedEdgeWithType dwe : graph.edgeSet()){
if(graph.getEdgeWeight(dwe) >= weightBoundary)
edgesOver.add(dwe);
}
return edgesOver;
}

private Set<WeightedVertex> getVertexSetFromEdges(List<WeightedEdgeWithType> edgeList){
Set<WeightedVertex> vertexSet = new HashSet<WeightedVertex>();
for(WeightedEdgeWithType dwe : edgeList){
vertexSet.add(graph.getEdgeSource(dwe));
vertexSet.add(graph.getEdgeTarget(dwe));
}
return vertexSet;
}

}
```

SubgraphExtractor.java

```
package extractor;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.jgrapht.GraphPath;
import org.jgrapht.alg.shortestpath.AllDirectedPaths;
```

```
import org.jgrapht.graph.DirectedWeightedMultigraph;
import org.jgrapht.graph.DirectedWeightedSubgraph;

/**
* Class used to generate the subgraph based on the threshold and hops set by the user.
*
* @author Danielle Rodriguez
*
*/
public class SubgraphExtractor {
private DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
private Set<WeightedVertex> chosenVertexList;
private Set<WeightedVertex> transitiveVertexList;
private List<GraphPath<WeightedVertex, WeightedEdgeWithType>> connectingPaths;
private double vertexThreshold, edgeThreshold;
private int hops;

/**
* Construct the Subgraph Extractor class with the base subgraph.
* @param graph - The main graph used as basis for the subgraph extractor.
*/
public SubgraphExtractor( DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph){
this.graph = graph;
chosenVertexList = new HashSet<>();
transitiveVertexList = new HashSet<>();
connectingPaths = new ArrayList<>();
vertexThreshold = Double.MAX_VALUE;
edgeThreshold = Double.MAX_VALUE;
hops = 1;
}

/**
* Get the Vertices that have been included in the required set of vertices in the latest generated subgraph.
* @return The list of Required vertices
*/
public Set<WeightedVertex> getChosenVertices(){
return chosenVertexList;
}

/**
* Get the vertices included in the graph that aren't part of the required set of vertices
* @return The list of Transitive Vertices
*/
public Set<WeightedVertex> getTransitiveVertices(){
return transitiveVertexList;
}

/**
* Get all paths in the generated subgraph
* @return The list of all paths in the subgraph
*/
public List<GraphPath<WeightedVertex, WeightedEdgeWithType>> getPathList(){
return connectingPaths;
}

/**
*
*
* @param isSkeleton
* @param addtnlVertices
* @return
*/
public DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> generateSubgraph(
boolean isSkeleton,
Set<WeightedVertex> addtnlVertices){
chosenVertexList.clear();
transitiveVertexList.clear();

VertexPicker vp = new VertexPicker(graph);
chosenVertexList = vp.getChosenVertices(vertexThreshold, edgeThreshold);
if(addtnlVertices != null)
chosenVertexList.addAll(addtnlVertices);
transitiveVertexList.addAll(chosenVertexList);

AllDirectedPaths<WeightedVertex, WeightedEdgeWithType> adp = new AllDirectedPaths<>(graph);
connectingPaths = adp.getAllPaths(chosenVertexList, chosenVertexList, true, hops);


Set<WeightedEdgeWithType> edgeList = new HashSet<WeightedEdgeWithType>();
for(GraphPath<WeightedVertex, WeightedEdgeWithType> curPath: connectingPaths){
if(curPath.getLength() == 0) continue;
if(curPath.getLength() == 1){
WeightedEdgeWithType curEdge = curPath.getEdgeList().get(0);
edgeList.add(curEdge);

} else if (!isSkeleton){
transitiveVertexList.addAll(curPath.getVertexList());
edgeList.addAll(curPath.getEdgeList());
}
}
DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> baseSubgraph =
new DirectedWeightedSubgraph<>(graph, transitiveVertexList, edgeList);
```

```
transitiveVertexList.removeAll(chosenVertexList);
return baseSubgraph;

}

/**
* Generate the subgraph with only the edge threshold, vertex thresholds, and number of hops
*
* @param vertexWeight - Threshold for the minimum weight of the required vertices
* @param edgeWeight - Threshold for the minimum weight of the edges to be included
* @param hops - Number of intransitive vertices in between required vertices
* @return The subgraph fulfilling the requirements passed
*/
public DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> generateSubgraph(double vertexThreshold, doub
this.vertexThreshold = vertexThreshold;
this.edgeThreshold = edgeThreshold;
this.hops = hops;
return generateSubgraph(false, null);
}

/**
* Generate the subgraph with only the edge threshold, vertex thresholds, number of hops, and whether or not the gra
*
* @param vertexWeight - Threshold for the minimum weight of the required vertices
* @param edgeWeight - Threshold for the minimum weight of the edges to be included
* @param hops - Number of intransitive vertices in between required vertices
* @param isSkeleton - Toggle if the returned subgraph should be only the required vertices
* @return The subgraph fulfilling the requirements passed
*/
public DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> generateSubgraph(double vertexWeight, double
this.vertexThreshold = vertexWeight;
this.edgeThreshold = edgeWeight;
this.hops = hops;
return generateSubgraph(isSkeleton, null);
}

/**
*
* @param vertexWeight
* @param edgeWeight
* @param hops
* @param isSkeleton
* @param addtnlVertices
* @return
*/
public DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> generateSubgraph(
double vertexWeight,
double edgeWeight,
int hops,
boolean isSkeleton,
Set<WeightedVertex> addtnlVertices){
this.vertexThreshold = vertexWeight;
this.edgeThreshold = edgeWeight;
this.hops = hops;
return generateSubgraph(isSkeleton, addtnlVertices);

}


public void setThreshold(double threshold){
vertexThreshold = threshold;
edgeThreshold = threshold;
}

public double getEdgeThreshold(){
return edgeThreshold;
}

public void clearSettings(){
vertexThreshold = Double.MAX_VALUE;
edgeThreshold = Double.MAX_VALUE;
hops = 1;
connectingPaths.clear();
}


public int getHops() {
return hops;
}

public void setHops(int hops) {
this.hops = hops;
}

}
```

DegreeCentralityComputer.java

```
package centrality;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;
```

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import org.jgrapht.graph.DirectedWeightedMultigraph;

/**
* Computes for the Indegree and Outdegree Centrality of a passed Directed Weighted graph.
*
* @author Danielle Rodriguez
*
*/
public class DegreeCentralityComputer {
private  DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
private Map<WeightedVertex, Double> inDegreeList;
private Map<WeightedVertex, Double> outDegreeList;

/**
* Construct an instance of the Degree Centrality Computer for the passed graph.
* @param graph - the input graph
*/
public DegreeCentralityComputer( DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph){
this.graph = graph;
setupDegreeDictionaries();
computeDegreeCentrality();
}

/**
* Calculate and return the In-degree Centrality values of the vertices in the graph.
* @return A map containing the vertices as the key and their corresponding in-degree centrality as the value
*/
public Map<WeightedVertex, Double> getInDegreeList(){
return inDegreeList;
}

/**
* Calculate and return the Out-degree Centrality values of the vertices in the graph.
* @return A map containing the vertices as the key and their corresponding out-degree centrality as the values
*/
public Map<WeightedVertex, Double> getOutDegreeList(){

return outDegreeList;
}

private void setupDegreeDictionaries(){
inDegreeList = new HashMap<>();
outDegreeList = new HashMap<>();
Set<WeightedVertex> vertexList = graph.vertexSet();
for(WeightedVertex vertex : vertexList){
inDegreeList.put(vertex, 0.0);
outDegreeList.put(vertex, 0.0);
}
}

private void computeDegreeCentrality(){
Set<WeightedVertex> vertexList = graph.vertexSet();
for(WeightedVertex vertex : vertexList){
inDegreeList.put(vertex, (double) graph.inDegreeOf(vertex));
outDegreeList.put(vertex, (double) graph.outDegreeOf(vertex));
}
inDegreeList = MapUtil.sortByValue(inDegreeList);
outDegreeList = MapUtil.sortByValue(outDegreeList);
}

}
```

ClosenessCentralityComputer.java

```
package centrality;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.util.HashMap;
import java.util.Map;

import org.jgrapht.alg.interfaces.ShortestPathAlgorithm.SingleSourcePaths;
import org.jgrapht.alg.shortestpath.FloydWarshallShortestPaths;
import org.jgrapht.graph.DirectedWeightedMultigraph;

/**
* Computes for the Closeness Centrality of a passed Directed Weighted graph.
*
* @author Danielle Rodriguez
*
*/
public class ClosenessCentralityComputer {
private DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
private Map<WeightedVertex, Double> closenessList;

/**
```

```
* Construct an instance of the Closeness Centrality Computer for the passed graph.
* @param graph - the input graph
*/
public ClosenessCentralityComputer(DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph){
this.graph = graph;
initializeClosenessList();
}

/**
*       Calculate and return the Closeness Centrality values of the vertices in the graph.
*  @return A map containing the vertices as the key and their corresponding closeness centrality as the value
*/
public Map<WeightedVertex, Double> getClosenessList(){
computeCloseness();
return MapUtil.sortByValue(closenessList);
}

private void initializeClosenessList() {
closenessList = new HashMap<>();
for(WeightedVertex wv : graph.vertexSet()){
closenessList.put(wv, 0.0);
}
}

/*
* Compute the closeness value of each vertex by getting the total
* length of the shortest paths to the vertex and using it to divide the number of nodes present
*/
private void computeCloseness(){
FloydWarshallShortestPaths<WeightedVertex, WeightedEdgeWithType> fwSP = new FloydWarshallShortestPaths<>(graph);
for(WeightedVertex vertex : graph.vertexSet()){
SingleSourcePaths<WeightedVertex, WeightedEdgeWithType> allShortestPaths = fwSP.getPaths(vertex);
double closenessVal = countPaths(allShortestPaths);
closenessList.put(vertex, closenessVal);
}
}

private double countPaths(SingleSourcePaths<WeightedVertex, WeightedEdgeWithType> asp){
double accumulatedPaths = 0;
for(WeightedVertex wv : graph.vertexSet()){
if(asp.getPath(wv) != null){
double pathWeight = asp.getPath(wv).getWeight();
if(pathWeight != 0)
accumulatedPaths += 1 / asp.getPath(wv).getWeight();
}
}
return accumulatedPaths;
}

}
```

BetweenessCentralityComputer.java

```
package centrality;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.util.ArrayList;
import java.util.Deque;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import org.jgrapht.alg.DirectedNeighborIndex;
import org.jgrapht.graph.DirectedWeightedMultigraph;

/**
* Computes for the Betweenness Centrality of a passed Directed Weighted graph.
*
* The class makes use of the algorithm proposed by Brandes (A Faster Algorithm for Betweenness Centrality, 2001).
*
* @author Danielle Rodriguez
*
*/
public class BetweenessCentralityComputer {
private  DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
private Map<WeightedVertex, Double> betweenessList;

/**
* Construct an instance of the Betweenness Centrality Computer for the passed graph.
* @param graph - the input graph
*/
public BetweenessCentralityComputer(DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph){
this.graph = graph;
initializeBetweenessList();
}

/**
* Calculate and return the Betweenness Centrality values of the vertices in the graph.
* @return A map containing the vertices as the key and their corresponding betweenness centrality as the value
```

```
*/
public Map<WeightedVertex, Double> getBetweenessList(){
computeBetweeness();
return MapUtil.sortByValue(betweenessList);
}

/**
* Initializes the Betweenness by populating it with the vertices and their centralities as zero.
*/
private void initializeBetweenessList() {
betweenessList = new HashMap<>();
for(WeightedVertex wv : graph.vertexSet()){
betweenessList.put(wv, 0.0);
}
}

/**
* Initialize the data for each iteration of the algorithm.
* @param dataList
*/
private void initializeBetweenessData(Map<WeightedVertex, BetweenessData> dataList){
for(WeightedVertex wv : graph.vertexSet()){
dataList.put(wv, new BetweenessData());
}
}

/**
* Compute for the Betweeness of each vertex in the graph.
*/
private void computeBetweeness(){
Map<WeightedVertex, BetweenessData> decor = new HashMap<>();
DirectedNeighborIndex<WeightedVertex, WeightedEdgeWithType> dni = new DirectedNeighborIndex<>(graph);

for(WeightedVertex s : graph.vertexSet()){
initializeBetweenessData(decor);
decor.get(s).shortestPathNum = 1;
decor.get(s).distance = 0;

Deque<WeightedVertex> S = new LinkedList<>();
Deque<WeightedVertex> Q = new LinkedList<>();
Q.addLast(s);

while(!Q.isEmpty()){
WeightedVertex v = Q.removeFirst();
S.addFirst(v);
for(WeightedVertex w : dni.successorsOf(v)){
if(decor.get(w).distance < 0){
Q.addLast(w);
decor.get(w).distance = decor.get(v).distance + 1;
}
if(decor.get(w).distance == decor.get(v).distance + 1){
decor.get(w).shortestPathNum += decor.get(v).shortestPathNum;
decor.get(w).predecessors.add(v);
}
}
}

while(!S.isEmpty()){
WeightedVertex w = S.removeFirst();
for(WeightedVertex v : decor.get(w).predecessors){
double pairDepFraction = decor.get(v).shortestPathNum / decor.get(w).shortestPathNum;
double neighborAddDep = decor.get(w).dependency + 1;
decor.get(v).dependency += pairDepFraction * neighborAddDep;
}
if(w != s)
betweenessList.put(w, betweenessList.get(w) + decor.get(w).dependency);
}
}
}

class BetweenessData{
List<WeightedVertex> predecessors;
double shortestPathNum;
double distance;
double dependency;

BetweenessData(){
predecessors = new ArrayList<>();
shortestPathNum = 0;
distance = -1;
dependency = 0;
}
}
}
```

EigenvectorCentraliltyComputer.java

```
package centrality;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.TreeSet;

import no.uib.cipr.matrix.DenseMatrix;
import no.uib.cipr.matrix.EVD;
import no.uib.cipr.matrix.NotConvergedException;

import org.jgrapht.graph.DirectedWeightedMultigraph;

/**
* Computes for the Eigenvector Centrality of a passed Directed Weighted graph.
*
* @author Danielle Rodriguez
*
*/
public class EigenvectorCentralityComputer {
private DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
private Map<WeightedVertex, Double> eigenvectorList;

/**
* Construct an instance of the Eigenvector Centrality Computer for the passed graph.
* @param graph - the input graph
*/
public EigenvectorCentralityComputer(DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph){
this.graph = graph;
initializeEigenvectorList();
}

/**
* Calculate and return the Eigenvector Centrality values of the vertices in the graph.
*
* @return A map containing the vertices as the key and their corresponding eigenvector centrality as the value
* @throws NotConvergedException It is be possible that the graph doesn't converge to an eigenvalue
*/
public Map<WeightedVertex, Double> getEigenvectorList() throws NotConvergedException{
computeEigenvector();
return MapUtil.sortByValue(eigenvectorList);
}

private void initializeEigenvectorList() {
eigenvectorList = new HashMap<>();
for(WeightedVertex wv : graph.vertexSet()){
eigenvectorList.put(wv, 0.0);
}
}

private void computeEigenvector() throws NotConvergedException{
DenseMatrix adjMatrix = new DenseMatrix(generateAdjacencyMatrix());
double[] maxEigenvect = getMaxEigenvector(adjMatrix);
List<WeightedVertex> vList = new ArrayList<WeightedVertex>(new TreeSet<WeightedVertex>(graph.vertexSet()));
int eigenCnt = 0;
for(WeightedVertex v : vList){
eigenvectorList.put(v, maxEigenvect[eigenCnt++]);
}
}

private double[] getMaxEigenvector(DenseMatrix adjMatrix) throws NotConvergedException {
EVD evd = new EVD(graph.vertexSet().size()).factor(adjMatrix);
double[] eigenVals = evd.getRealEigenvalues();
int maxIndex = 0;
for(int i = 0; i < eigenVals.length; i++){
if(Math.abs(eigenVals[i]) > Math.abs(eigenVals[maxIndex]))
maxIndex = i;
}
DenseMatrix eigenVectorsR = evd.getRightEigenvectors();
double[] principalEigenVector = getEigenVector(eigenVectorsR, maxIndex);

return principalEigenVector;
}

private double[] getEigenVector(DenseMatrix eigenVectors, int maxIndex) {
int size = eigenVectors.numRows();
double[] eigenVect = new double[size];
for(int i = 0; i < size; i++){
eigenVect[i] = Math.abs(eigenVectors.get(i, maxIndex));
}
return eigenVect;
}

private double[][] generateAdjacencyMatrix(){
List<WeightedVertex> vList = new ArrayList<WeightedVertex>(new TreeSet<WeightedVertex>(graph.vertexSet()));
double[][] adjMat = generateEmptyMatrix(vList.size());
int row = 0, col = 0;
for(WeightedVertex source : vList){
for(WeightedEdgeWithType edge : graph.edgesOf(source)){
col = vList.indexOf(graph.getEdgeTarget(edge));
adjMat[row][col] = 1;
}
```

```
row++;
}
return adjMat;
}

private double[][] generateEmptyMatrix(int size){
double[][] array = new double[size][size];
for (double[]a : array) {
Arrays.fill(a, 0);
}
for (int i=0; i<size; i++) array[i][i] = 1;
return array;
}

}
```

MapUtil.java

```
package centrality;

import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

class MapUtil {
public static <K, V extends Comparable<? super V>> Map<K, V>
sortByValue( Map<K, V> map )
{
List<Map.Entry<K, V>> list =
new LinkedList<Map.Entry<K, V>>( map.entrySet() );
Collections.sort( list, new Comparator<Map.Entry<K, V>>()
{
public int compare( Map.Entry<K, V> o1, Map.Entry<K, V> o2 )
{
return (o1.getValue()).compareTo( o2.getValue() );
}
} );

Map<K, V> result = new LinkedHashMap<K, V>();
for (Map.Entry<K, V> entry : list)
{
result.put( entry.getKey(), entry.getValue() );
}
return result;
}
}
```

GraphFileReader.java

```
package files;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.io.FileReader;
import java.io.IOException;

import org.jgrapht.graph.DirectedWeightedMultigraph;

import com.opencsv.CSVReader;

/**
* Class for reading the .csv file containing the graph.
* Returns a Graph Object of the file after reading it.
* @author Danielle Rodriguez
*
*/
public class GraphFileReader {
private FileReader file;
DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;

/**
* Constructor accepting the file containing the graph
* @param file - File to be read.
*/
public GraphFileReader(FileReader file){
this.file = file;
}

/**
* Reads the contents of the file and returns the Object containing the graph
* @return Object that is the graphical representation of the contents of the file
* @throws IOException
*/
public  DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> generateGraphFromFile() throws IOException

graph = new DirectedWeightedMultigraph<>(WeightedEdgeWithType.class);
CSVReader reader = new CSVReader(file);
String[] nextLine;
while ((nextLine = reader.readNext()) != null) {
```

```
addToGraph(nextLine);
}

reader.close();

return graph;
}

/*
* Adds a new edge to the graph since each edge is formatted as:
* node_1, node_2, node_1_type, node_2_type, edge_type, edge_weight
*/
private void addToGraph(String[] lineVals){
WeightedVertex source = new WeightedVertex(lineVals[0].replaceAll("\\s+",""), lineVals[2].replaceAll("\\s+",""));
graph.addVertex(source);
WeightedVertex dest = new WeightedVertex(lineVals[1].replaceAll("\\s+",""), lineVals[3].replaceAll("\\s+",""));
graph.addVertex(dest);

WeightedEdgeWithType edge = graph.addEdge(source, dest);
edge.setType(lineVals[4].replaceAll("\\s+",""));
graph.setEdgeWeight(edge, Double.parseDouble(lineVals[5]));
}

}
```

GraphFileWriter.java

```
package files;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.io.FileWriter;
import java.io.IOException;

import org.jgrapht.graph.DirectedWeightedSubgraph;

import com.opencsv.CSVWriter;

/**
* Class for writing the .csv file that will contain the subgraph generated in the program
*
* @author Danielle Rodriguez
*
*/
public class GraphFileWriter {
private FileWriter file;

public GraphFileWriter(FileWriter file){
this.file = file;
}

public void writeSubgraphToFile(DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> subgraph) throws IOE
CSVWriter writer = new CSVWriter(file);
for(WeightedEdgeWithType edge : subgraph.edgeSet()){
String[] details = getEdgeDetails(subgraph, edge);
writer.writeNext(details);
}
writer.close();
}

private String[] getEdgeDetails(DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> subgraph, WeightedEd
WeightedVertex source = subgraph.getEdgeSource(wEdge);
WeightedVertex target = subgraph.getEdgeTarget(wEdge);
String[] list = {
source.getName(),
target.getName(),
source.getType(),
target.getType(),
wEdge.getType(),
Double.toString(subgraph.getEdgeWeight(wEdge))
};
return list;
}
}
```

GraphPanel.java

```
package window.views;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Map;
import java.util.Set;

import javax.swing.JLabel;
import javax.swing.JPanel;
```

```
import org.jgrapht.ext.JGraphXAdapter;
import org.jgrapht.graph.DirectedWeightedMultigraph;

import com.mxgraph.layout.mxFastOrganicLayout;
import com.mxgraph.layout.mxParallelEdgeLayout;
import com.mxgraph.model.mxCell;
import com.mxgraph.model.mxICell;
import com.mxgraph.swing.mxGraphComponent;

/**
* Panel that contains the graph that is loaded into the tool.
* @author Einad
*
*/
@SuppressWarnings("serial")
public class GraphPanel extends JPanel {

private DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter;
mxGraphComponent graphCom;
mxCell selectedCell;
double scale;

/**
* Create the panel.
*/
public GraphPanel (DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph) {
this.graph = graph;
//setBorder(null);
setBackground(Color.green);
showGraph();
}

public JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> getGraphAdapter(){
return graphAdapter;
}

public mxGraphComponent getGraphComponent(){
return graphCom;
}

/**
* Places the loaded graph into a JPanel
*/
public void showGraph(){
graphAdapter = new JGraphXAdapter<>(graph);
WindowUtils.setupGraph(graphAdapter);
mxFastOrganicLayout organicLayout = new mxFastOrganicLayout(graphAdapter);
mxParallelEdgeLayout parallelLayout = new mxParallelEdgeLayout(graphAdapter);
organicLayout.setForceConstant(150); // the higher, the more separated
organicLayout.execute(graphAdapter.getDefaultParent());
parallelLayout.execute(graphAdapter.getDefaultParent());

graphCom = new mxGraphComponent(graphAdapter);
graphCom.setConnectable(false);
this.setLayout(new BorderLayout());
this.add(graphCom, BorderLayout.CENTER);
}

public void colorVertices(Set<WeightedVertex> verticesToBeColored, String color){
Map<WeightedVertex, mxICell> vertexTable = graphAdapter.getVertexToCellMap();
Object[] mxCell = new Object[verticesToBeColored.size()];
int i = 0;

for(WeightedVertex cv : verticesToBeColored){
mxICell cell = vertexTable.get(cv);
mxCell[i++] = (Object) cell;
}

graphAdapter.setCellStyle("fillColor="+color, mxCell);

}

void setGraphListener(final JPanel elementInfoContentP){
final mxGraphComponent graphComponent = this.getGraphComponent();
graphComponent.getGraphControl().addMouseListener(new MouseAdapter(){
@Override
public void mouseReleased(MouseEvent e){
Object cell = graphComponent.getCellAt(e.getX(), e.getY());

if (cell != null){
selectedCell = (mxCell) cell;
StringBuilder sb = new StringBuilder();
try{
WeightedVertex wv = (WeightedVertex) selectedCell.getValue();
elementInfoContentP.removeAll();
sb.append("Name = ").append(wv.getName());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Type = ").append(wv.getType());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
```

```
sb.append("Weight = ").append(wv.getWeight());
elementInfoContentP.add(new JLabel(sb.toString()));
elementInfoContentP.revalidate();
elementInfoContentP.repaint();


} catch(ClassCastException cce){
WeightedEdgeWithType wEdge = (WeightedEdgeWithType)selectedCell.getValue();
elementInfoContentP.removeAll();
sb.append("Source = ").append(graph.getEdgeSource(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Target = ").append(graph.getEdgeTarget(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Type = ").append(wEdge.getType());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Weight = ").append(graph.getEdgeWeight(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
elementInfoContentP.revalidate();
elementInfoContentP.repaint();

}
} else {
elementInfoContentP.removeAll();
elementInfoContentP.revalidate();
elementInfoContentP.repaint();
}
}
});
}

void zoomGraph(boolean isZoomIn){
if(isZoomIn){
graphCom.zoomIn();
} else{
graphCom.zoomOut();
}
}

mxCell getSelectedCell(){
return selectedCell;
}

}
```

SubgraphWindow.java

```
package window.views;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FileDialog;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.border.LineBorder;

import org.jgrapht.GraphPath;
import org.jgrapht.ext.JGraphXAdapter;
import org.jgrapht.graph.DirectedWeightedMultigraph;
import org.jgrapht.graph.DirectedWeightedSubgraph;

import window.views.utils.UserConstants;

import com.mxgraph.layout.mxFastOrganicLayout;
import com.mxgraph.layout.mxParallelEdgeLayout;
import com.mxgraph.model.mxCell;
import com.mxgraph.model.mxICell;
import com.mxgraph.swing.mxGraphComponent;

import extractor.SubgraphExtractor;
import files.GraphFileWriter;
import graph.WeightedEdgeWithType;
```

```
import graph.WeightedVertex;

@SuppressWarnings("serial")
public class SubgraphWindow extends JFrame {
private SubgraphExtractor se;
DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> completeSubG;
JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter;
private JPanel elementInfoContentP, subgraphInfoContentP;
private JPanel contentPane;

SubgraphExtractor getExtractor(){
return se;
}

Set<WeightedVertex> getChosenVertices(){
return se.getChosenVertices();
}

/**
* Create the frame.
*/
public SubgraphWindow(DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph){
this.graph = graph;
setTitle("Generated Subgraph");
setResizable(false);
setBounds(100, 100, 700, 500);
contentPane = new JPanel();
contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
setContentPane(contentPane);
contentPane.setLayout(null);

JPanel elementInfoP = new JPanel();
elementInfoP.setBounds(457, 124, 217, 258);
elementInfoP.setBorder(new LineBorder(Color.decode("#808B96")));
contentPane.add(elementInfoP);
elementInfoP.setLayout(null);

JPanel elementInfoLabelP = new JPanel();
elementInfoLabelP.setBackground(Color.decode("#808B96"));
elementInfoLabelP.setBounds(0, 0, 217, 23);
elementInfoP.add(elementInfoLabelP);
elementInfoLabelP.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));

JLabel elementInfoL = new JLabel("Element Information");
elementInfoL.setFont(new Font("Tahoma", Font.BOLD, 12));
elementInfoLabelP.add(elementInfoL);

elementInfoContentP = new JPanel();
elementInfoContentP.setBounds(0, 22, 217, 236);
elementInfoContentP.setLayout(new BoxLayout(elementInfoContentP, BoxLayout.Y_AXIS));
elementInfoP.add(elementInfoContentP);

JButton closeB = new JButton("Close");
closeB.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
setVisible(false);
dispose();
}
});
closeB.setBounds(475, 410, 89, 23);
WindowUtils.setButtonStyle(closeB);
contentPane.add(closeB);

JButton exportB = new JButton("Export");
final JFrame main = this;
exportB.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
FileDialog fd = new FileDialog(main, "Save Subgraph...", FileDialog.SAVE);
fd.setVisible(true);

String fileName = fd.getDirectory() + fd.getFile();
try {
GraphFileWriter gfw = new GraphFileWriter(new FileWriter(fileName));
gfw.writeSubgraphToFile(completeSubG);
} catch (IOException IOe) {
IOe.printStackTrace();
}
}
});
exportB.setBounds(574, 410, 89, 23);
WindowUtils.setButtonStyle(exportB);
contentPane.add(exportB);

JPanel subgraphInfoP = new JPanel();
subgraphInfoP.setBounds(457, 11, 217, 102);
subgraphInfoP.setBorder(new LineBorder(Color.decode("#808B96")));
contentPane.add(subgraphInfoP);
subgraphInfoP.setLayout(null);

JPanel subgraphInfoLabelP = new JPanel();
subgraphInfoLabelP.setBounds(0, 0, 217, 24);
```

```
subgraphInfoLabelP.setBackground(Color.decode("#808B96"));
subgraphInfoP.add(subgraphInfoLabelP);
subgraphInfoLabelP.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));

JLabel subgraphInfoL = new JLabel("Subgraph Information");
subgraphInfoL.setFont(new Font("Tahoma", Font.BOLD, 12));
subgraphInfoLabelP.add(subgraphInfoL);

subgraphInfoContentP = new JPanel();
subgraphInfoContentP.setBounds(0, 22, 217, 80);
subgraphInfoContentP.setLayout(new BoxLayout(subgraphInfoContentP, BoxLayout.Y_AXIS));
subgraphInfoP.add(subgraphInfoContentP);
}

void generateSubgraph(SubgraphExtractor se, Set<WeightedVertex> addVert){
this.se = se;
completeSubG = se.generateSubgraph(false, addVert);
DirectedWeightedSubgraph<WeightedVertex, WeightedEdgeWithType> subG = se.generateSubgraph(true, addVert);
if(se.getEdgeThreshold() == Double.MAX_VALUE)
subgraphInfoContentP.add(new JLabel("Threshold: n/a"));
else
subgraphInfoContentP.add(new JLabel("Threshold:  " + se.getEdgeThreshold()));
subgraphInfoContentP.add(new JLabel("Hop Number: " + se.getHops()));

graphAdapter = new JGraphXAdapter<>(subG);
graphAdapter.getModel().beginUpdate();
try {
hideEdges(graphAdapter);
} finally {
graphAdapter.getModel().endUpdate();
}

Map<WeightedVertex, mxICell> vertexTable = graphAdapter.getVertexToCellMap();
Set<WeightedVertex> chosen = se.getChosenVertices();

Object[] mxCell = new Object[chosen.size()];
int i = 0;
for(WeightedVertex cv : chosen){
mxICell cell = vertexTable.get(cv);
mxCell[i++] = (Object) cell;
}

graphAdapter.setCellStyle("fillColor="+UserConstants.CHOSEN_COLOR, mxCell);
WindowUtils.setupGraph(graphAdapter);
mxGraphComponent graphCom = new mxGraphComponent(graphAdapter);
graphCom.setConnectable(false);
setGraphMouseListener(graphCom);

mxFastOrganicLayout organicLayout = new mxFastOrganicLayout(graphAdapter);
mxParallelEdgeLayout parallelLayout = new mxParallelEdgeLayout(graphAdapter);

organicLayout.setForceConstant(150); // the higher, the more separated
organicLayout.execute(graphAdapter.getDefaultParent());
parallelLayout.execute(graphAdapter.getDefaultParent());

JPanel subgraphP = new JPanel();
subgraphP.setBounds(10, 11, 437, 439);
subgraphP.setLayout(new BorderLayout());
subgraphP.add(graphCom);
contentPane.add(subgraphP);

}

private void setGraphMouseListener(final mxGraphComponent graphCom) {
graphCom.getGraphControl().addMouseListener(new MouseAdapter()
{

public void mouseReleased(MouseEvent e)
{
Object cell = graphCom.getCellAt(e.getX(), e.getY());
mxCell cellM = (mxCell) cell;
mxICell parent = null;
if(cellM != null && !isTransitiveEdge(cellM)){
parent = cellM.getParent();
StringBuilder sb = new StringBuilder();
try{
WeightedVertex wv = (WeightedVertex) cellM.getValue();


elementInfoContentP.removeAll();
sb.append("Name = ").append(wv.getName());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Type = ").append(wv.getType());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Weight = ").append(wv.getWeight());
elementInfoContentP.add(new JLabel(sb.toString()));
elementInfoContentP.revalidate();
elementInfoContentP.repaint();
```

```
} catch(ClassCastException cce){
WeightedEdgeWithType wEdge = (WeightedEdgeWithType)cellM.getValue();
elementInfoContentP.removeAll();
sb.append("Source = ").append(graph.getEdgeSource(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Target = ").append(graph.getEdgeTarget(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Type = ").append(wEdge.getType());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Weight = ").append(graph.getEdgeWeight(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
elementInfoContentP.revalidate();
elementInfoContentP.repaint();

}
} else {
elementInfoContentP.removeAll();
elementInfoContentP.revalidate();
elementInfoContentP.repaint();
}
if(e.getClickCount() == 1){
if (cell != null && isTransitiveEdge(cellM))
{
if(cellM.isCollapsed()){
graphAdapter.foldCells(false, false, new Object[]{cell});
toggleElements(cellM, true, e);
graphAdapter.setCellStyle("strokeColor="+UserConstants.SUBGRAPH_BACKGROUND+";fontColor="+UserConstants.SUBGRAPH_BACK

}
}

} else if(e.getClickCount() == 2){
if(cell != null && parent != graphAdapter.getDefaultParent()){
if(!cellM.isCollapsed()){
graphAdapter.foldCells(true, false, new Object[]{parent});
toggleElements((mxCell)parent,false, e);
graphAdapter.setCellStyle("dashed=true;strokeColor="+UserConstants.INDIRECT_COLOR+";fontColor="+UserConstants.INDIRE
}
}
}
}
});
}

private boolean isTransitiveEdge(mxCell edge){
return edge.isEdge() && edge.getChildCount() > 0;
}

private void toggleElements(mxCell cell, boolean visible, MouseEvent e){
double INIT_LOC = 2.0;
int numOfChild = cell.getChildCount();
Set<Object> cellList = new HashSet<>();
double loc = 0;
for(int i = 0; i < numOfChild; i++){
mxICell child = cell.getChildAt(i);
cellList.add((Object)child);
loc = graphAdapter.getCellGeometry(child).getX() + graphAdapter.getCellGeometry(child).getY();
child.setVisible(visible);

int numOfEdges = child.getEdgeCount();
for(int j = 0; j < numOfEdges; j++){
mxICell edge = child.getEdgeAt(j);
edge.setVisible(visible);
}
}
if(visible && loc==INIT_LOC){
graphAdapter.moveCells(cellList.toArray(),e.getX(),e.getY());
}
}

private void hideEdges(JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter){
List<GraphPath<WeightedVertex, WeightedEdgeWithType>> connectingPaths = se.getPathList();
Map<WeightedVertex, mxICell> vertexTable = graphAdapter.getVertexToCellMap();
for(GraphPath<WeightedVertex, WeightedEdgeWithType> curPath : connectingPaths){
if(curPath.getLength() > 1 && doesNotContainSelected(curPath)){
Object mainParent = insertParentEdge(
vertexTable.get(curPath.getStartVertex()),
vertexTable.get(curPath.getEndVertex()),
getPathWeight(curPath));
Object[] cells = generateGroupObjects(curPath, mainParent, vertexTable);
graphAdapter.groupCells(mainParent, 1, cells);
graphAdapter.foldCells(true, false, new Object[]{mainParent});
graphAdapter.setCellStyle("dashed=true;strokeColor="+UserConstants.INDIRECT_COLOR+";fillColor="+UserConstants.INDIRE
graphAdapter.setCellStyle("dashed=true;strokeColor="+UserConstants.INDIRECT_COLOR+";fontColor="+UserConstants.INDIRE
}
}
}

private double getPathWeight(GraphPath<WeightedVertex, WeightedEdgeWithType> path){
```

```
List<WeightedEdgeWithType> edgeList = path.getEdgeList();
double weight = 0.0;
for(WeightedEdgeWithType edge : edgeList)
weight += graph.getEdgeWeight(edge);
return weight;
}

private Object[] generateGroupObjects(
GraphPath<WeightedVertex, WeightedEdgeWithType> curPath,
Object mainParent,
Map<WeightedVertex, mxICell> vertexTable) {
List<WeightedEdgeWithType> edgePath = curPath.getEdgeList();
List<WeightedVertex> edgeVertex = curPath.getVertexList();
List<Object> mxCellAr = new ArrayList<>(1);
List<Object> mxVertAr = new ArrayList<>();
for(int i = 0; i < edgeVertex.size(); i++){
WeightedVertex curVert = edgeVertex.get(i);
if(i == 0 || i == edgeVertex.size()-1){
mxVertAr.add((Object) vertexTable.get(curVert));
}else{
Object vInGrp = graphAdapter.
insertVertex(mainParent, curVert.getName(), curVert, 0, 0, 40, 20, "fillColor="+UserConstants.CHOSEN_COLOR);
mxVertAr.add(vInGrp);
mxCellAr.add(vInGrp);
((mxCell)vInGrp).setVisible(false);
}
}
int curVertex = 0;
for(WeightedEdgeWithType wew : edgePath){
Object e = graphAdapter.insertEdge(mainParent, null, wew, mxVertAr.get(curVertex), mxVertAr.get(++curVertex));
mxCellAr.add(e);
((mxCell)e).setVisible(false);
}
return mxCellAr.toArray();
}

private boolean doesNotContainSelected(GraphPath<WeightedVertex, WeightedEdgeWithType> curPath){
List<WeightedVertex> vertexList = curPath.getVertexList();
for(int i = 1; i < vertexList.size()-1; i++){ //skip the first and last vertices
if(se.getChosenVertices().contains(vertexList.get(i)))
return false;
}
return true;
}

private Object insertParentEdge(mxICell source, mxICell target, double totalWeight){
return graphAdapter.insertEdge(graphAdapter.getDefaultParent(), "e"+source+target, totalWeight, source, target);
}
}
```

HopsWindow.java

```
package window.views;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;

import org.jgrapht.ext.JGraphXAdapter;

import com.mxgraph.layout.mxCircleLayout;
import com.mxgraph.layout.mxParallelEdgeLayout;
import com.mxgraph.swing.mxGraphComponent;

import java.awt.Font;

@SuppressWarnings("serial")
class HopsWindow extends JFrame{
JSlider thresholdSlider;
JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter;
boolean hasSubmitted = false;
int hops;

public HopsWindow(JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter) {
this.graphAdapter = graphAdapter;
initialize();
}

private void initialize() {
setTitle("Generate Subgraph...");
setResizable(false);
setBounds(100, 100, 477, 592);
```

```
getContentPane().setLayout(null);
setLocationRelativeTo(null);

JPanel subgraphP = new JPanel();
subgraphP.setBounds(10, 11, 437, 439);
subgraphP.setLayout(new BorderLayout());
WindowUtils.setupGraph(graphAdapter);
mxCircleLayout organicLayout = new mxCircleLayout(graphAdapter);
mxParallelEdgeLayout parallelLayout = new mxParallelEdgeLayout(graphAdapter);
organicLayout.execute(graphAdapter.getDefaultParent());
parallelLayout.execute(graphAdapter.getDefaultParent());

mxGraphComponent graphCom = new mxGraphComponent(graphAdapter);
graphCom.setConnectable(false);
subgraphP.add(graphCom, BorderLayout.CENTER);
getContentPane().add(subgraphP);

final JSlider hopSl = createHopsSlider();
hopSl.setBounds(104, 491, 257, 51);
getContentPane().add(hopSl);

JLabel lblNumberOfHops = new JLabel("Number of Hops");
lblNumberOfHops.setBounds(104, 460, 98, 14);
getContentPane().add(lblNumberOfHops);

JButton setValuesB = new JButton("Generate");
setValuesB.setFont(new Font("Verdana", Font.PLAIN, 9));
setValuesB.addActionListener(new ActionListener(){
@Override
public void actionPerformed(ActionEvent e) {
hops = hopSl.getValue();
hasSubmitted = true;
dispatchEvent(new WindowEvent(HopsWindow.this, WindowEvent.WINDOW_CLOSING));
}

});
setValuesB.setBounds(254, 457, 107, 23);
WindowUtils.setButtonStyle(setValuesB);
getContentPane().add(setValuesB);
}

private JSlider createHopsSlider(){
JSlider slider = new JSlider(JSlider.HORIZONTAL, 1, 3, 1);

slider.setMinorTickSpacing(1);
slider.setPaintTicks(true);
slider.setPaintLabels(true);

slider.setLabelTable(slider.createStandardLabels(1));
return slider;
}

public boolean hasSubmitted(){
return hasSubmitted;
}

public int getHops(){
return hops;
}
}
```

ThresholdWindow.java

```
package window.views;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.util.Hashtable;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

@SuppressWarnings("serial")
class ThresholdWindow extends JFrame{
private JTextField thresholdTf;
JSlider thresholdSlider;
boolean hasSubmitted = false;
int threshold;
int thresholdLim;

public ThresholdWindow(int thresholdLim, int threshold) {
this.thresholdLim = thresholdLim;
if(threshold == Double.MAX_VALUE)
threshold = 0;

this.threshold = threshold;
```

```
initialize();
}

private void initialize() {
setTitle("Set Threshold...");
setResizable(false);
setBounds(100, 100, 293, 150);
getContentPane().setLayout(null);
setLocationRelativeTo(null);

JLabel lblThreshold = new JLabel("Threshold");
lblThreshold.setBounds(10, 15, 76, 14);
getContentPane().add(lblThreshold);

thresholdTf = new JTextField("1");
thresholdTf.setBounds(72, 12, 76, 20);
thresholdTf.setEditable(false);

getContentPane().add(thresholdTf);
thresholdTf.setColumns(10);

JButton setValuesB = new JButton("Set Values");
setValuesB.addActionListener(new ActionListener(){
@Override
public void actionPerformed(ActionEvent e) {
threshold = Integer.parseInt(thresholdTf.getText());
hasSubmitted = true;
dispatchEvent(new WindowEvent(ThresholdWindow.this, WindowEvent.WINDOW_CLOSING));
}

});
setValuesB.setBounds(166, 11, 101, 23);
WindowUtils.setButtonStyle(setValuesB);
getContentPane().add(setValuesB);

thresholdSlider = createThresholdSlider();
thresholdSlider.setBounds(10, 47, 257, 51);
getContentPane().add(thresholdSlider);
}

private JSlider createThresholdSlider(){
final JSlider slider = new JSlider(JSlider.HORIZONTAL, 1, thresholdLim, 1);

Hashtable<Integer, JLabel> labelT = new Hashtable<>();
labelT.put(new Integer(1), new JLabel("1"));
labelT.put(new Integer(thresholdLim), new JLabel(Integer.toString(thresholdLim)));
slider.setLabelTable(labelT);

slider.addChangeListener(
new ChangeListener(){
public void stateChanged( ChangeEvent e ){
thresholdTf.setText(Integer.toString(slider.getValue()));
}
}
);
slider.setPaintLabels(true);
slider.setValue(threshold);
return slider;
}

public boolean hasSubmitted(){
return hasSubmitted;
}

public int getThreshold(){
return threshold;
}
}
```

GraphPanel.java

```
package window.views;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Map;
import java.util.Set;

import javax.swing.JLabel;
import javax.swing.JPanel;

import org.jgrapht.ext.JGraphXAdapter;
import org.jgrapht.graph.DirectedWeightedMultigraph;

import com.mxgraph.layout.mxFastOrganicLayout;
import com.mxgraph.layout.mxParallelEdgeLayout;
import com.mxgraph.model.mxCell;
```

```
import com.mxgraph.model.mxICell;
import com.mxgraph.swing.mxGraphComponent;

/**
* Panel that contains the graph that is loaded into the tool.
* @author Einad
*
*/
@SuppressWarnings("serial")
public class GraphPanel extends JPanel {

private DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter;
mxGraphComponent graphCom;
mxCell selectedCell;
double scale;

/**
* Create the panel.
*/
public GraphPanel (DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph) {
this.graph = graph;
//setBorder(null);
setBackground(Color.green);
showGraph();
}

public JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> getGraphAdapter(){
return graphAdapter;
}

public mxGraphComponent getGraphComponent(){
return graphCom;
}

/**
* Places the loaded graph into a JPanel
*/
public void showGraph(){
graphAdapter = new JGraphXAdapter<>(graph);
WindowUtils.setupGraph(graphAdapter);
mxFastOrganicLayout organicLayout = new mxFastOrganicLayout(graphAdapter);
mxParallelEdgeLayout parallelLayout = new mxParallelEdgeLayout(graphAdapter);
organicLayout.setForceConstant(150); // the higher, the more separated
organicLayout.execute(graphAdapter.getDefaultParent());
parallelLayout.execute(graphAdapter.getDefaultParent());

graphCom = new mxGraphComponent(graphAdapter);
graphCom.setConnectable(false);
this.setLayout(new BorderLayout());
this.add(graphCom, BorderLayout.CENTER);
}

public void colorVertices(Set<WeightedVertex> verticesToBeColored, String color){
Map<WeightedVertex, mxICell> vertexTable = graphAdapter.getVertexToCellMap();
Object[] mxCell = new Object[verticesToBeColored.size()];
int i = 0;

for(WeightedVertex cv : verticesToBeColored){
mxICell cell = vertexTable.get(cv);
mxCell[i++] = (Object) cell;
}

graphAdapter.setCellStyle("fillColor="+color, mxCell);

}

void setGraphListener(final JPanel elementInfoContentP){
final mxGraphComponent graphComponent = this.getGraphComponent();
graphComponent.getGraphControl().addMouseListener(new MouseAdapter(){
@Override
public void mouseReleased(MouseEvent e){
Object cell = graphComponent.getCellAt(e.getX(), e.getY());

if (cell != null){
selectedCell = (mxCell) cell;
StringBuilder sb = new StringBuilder();
try{
WeightedVertex wv = (WeightedVertex) selectedCell.getValue();
elementInfoContentP.removeAll();
sb.append("Name = ").append(wv.getName());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Type = ").append(wv.getType());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Weight = ").append(wv.getWeight());
elementInfoContentP.add(new JLabel(sb.toString()));
elementInfoContentP.revalidate();
elementInfoContentP.repaint();
```

```
} catch(ClassCastException cce){
WeightedEdgeWithType wEdge = (WeightedEdgeWithType)selectedCell.getValue();
elementInfoContentP.removeAll();
sb.append("Source = ").append(graph.getEdgeSource(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Target = ").append(graph.getEdgeTarget(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Type = ").append(wEdge.getType());
elementInfoContentP.add(new JLabel(sb.toString()));
sb.setLength(0);
sb.append("Weight = ").append(graph.getEdgeWeight(wEdge));
elementInfoContentP.add(new JLabel(sb.toString()));
elementInfoContentP.revalidate();
elementInfoContentP.repaint();

}
} else {
elementInfoContentP.removeAll();
elementInfoContentP.revalidate();
elementInfoContentP.repaint();
}
}
});
}

void zoomGraph(boolean isZoomIn){
if(isZoomIn){
graphCom.zoomIn();
} else{
graphCom.zoomOut();
}
}

mxCell getSelectedCell(){
return selectedCell;
}

}
```

OpeningWindow.java

```
package window.views;
import java.awt.BorderLayout;
import java.awt.Color;

import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JWindow;


@SuppressWarnings("serial")
class OpeningWindow extends JWindow {
private int duration;
public OpeningWindow(int d) {
duration = d;
}

public void showSplash() {
JPanel content = (JPanel)getContentPane();
content.setBackground(Color.white);

setBounds(100, 100, 768, 275);
setLocationRelativeTo(null);

JLabel label = new JLabel(new ImageIcon("resources/MenuBg.png"));
content.add(label, BorderLayout.CENTER);

setVisible(true);

try { Thread.sleep(duration); } catch (Exception e) {}

setVisible(false);
}
}
```

MainWindow.java

```
package window.views;

import extractor.SubgraphExtractor;
import extractor.VertexPicker;
import files.GraphFileReader;
import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.awt.Color;
import java.awt.EventQueue;
import java.awt.FileDialog;
import java.awt.FlowLayout;
import java.awt.Font;
```

```
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

import javax.imageio.ImageIO;
import javax.swing.BoxLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.UIManager;
import javax.swing.border.LineBorder;

import no.uib.cipr.matrix.NotConvergedException;

import org.jgrapht.ext.JGraphXAdapter;
import org.jgrapht.graph.DirectedWeightedMultigraph;

import window.views.utils.UserConstants;
import centrality.BetweenessCentralityComputer;
import centrality.ClosenessCentralityComputer;
import centrality.DegreeCentralityComputer;
import centrality.EigenvectorCentralityComputer;

import com.mxgraph.model.mxCell;
import com.mxgraph.util.mxCellRenderer;

public class MainWindow{
private JFrame frmNetExtranalysis;
private GraphPanel graphP;
private DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph;
private Set<WeightedVertex> addVertices;
private SubgraphExtractor se;
JPanel graphCentralityP, centLblP, centralityLegendP, elementInfoContentP;
JPanel subgraphInfoContentP;


public static void main(String[] args) {
OpeningWindow open = new OpeningWindow(3500);
open.showSplash();
EventQueue.invokeLater(new Runnable() {
public void run() {
try {
MainWindow window = new MainWindow();
window.frmNetExtranalysis.setVisible(true);
} catch (Exception e) {
e.printStackTrace();
}
}
});
}

public MainWindow() {
initialize();
}

private void initialize() {
addVertices = new TreeSet<>();

ImageIcon img = new ImageIcon("resources/bigLogo.png");

frmNetExtranalysis = new JFrame();
frmNetExtranalysis.setIconImage(img.getImage());
frmNetExtranalysis.setTitle("Biological Net Extralysis: A Biological Network Analysis and Subnetwork Extraction Too
frmNetExtranalysis.setBounds(100, 100, 900, 600);
frmNetExtranalysis.setLocationRelativeTo(null);
frmNetExtranalysis.setResizable(false);
frmNetExtranalysis.getContentPane().setBackground(UIManager.getColor("Panel.background"));
frmNetExtranalysis.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frmNetExtranalysis.getContentPane().setLayout(null);
```

```
JPanel elelementInfoP = new JPanel();
elelementInfoP.setBorder(new LineBorder(Color.decode("#808B96")));
elelementInfoP.setBounds(578, 11, 296, 144);
frmNetExtranalysis.getContentPane().add(elelementInfoP);
elelementInfoP.setLayout(null);

JPanel elementInfoLblP = new JPanel();
elementInfoLblP.setBackground(Color.decode("#808B96"));
elementInfoLblP.setBounds(0, 0, 296, 23);
elelementInfoP.add(elementInfoLblP);
elementInfoLblP.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel elementInfoL = new JLabel("Element Information");
elementInfoL.setFont(new Font("Tahoma", Font.BOLD, 12));
elementInfoLblP.add(elementInfoL);

elementInfoContentP = new JPanel();
elementInfoContentP.setBounds(5, 23, 286, 93);
elementInfoContentP.setLayout(new BoxLayout(elementInfoContentP, BoxLayout.Y_AXIS));
elelementInfoP.add(elementInfoContentP);

JButton addElemB = new JButton("Add Element");
addElemB.setFont(new Font("Tahoma", Font.PLAIN, 10));
addElemB.setBounds(193, 117, 93, 23);
WindowUtils.setButtonStyle(addElemB);
elelementInfoP.add(addElemB);

addElemB.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
mxCell selectedCell = graphP.getSelectedCell();
try{
WeightedVertex selectedV = (WeightedVertex) selectedCell.getValue();
addVertices.add(selectedV);
} catch (ClassCastException cce){
WeightedEdgeWithType selectedE = (WeightedEdgeWithType) selectedCell.getValue();
addVertices.add(graph.getEdgeSource(selectedE));
addVertices.add(graph.getEdgeTarget(selectedE));
}
graphP.colorVertices(addVertices,UserConstants.CHOSEN_COLOR);

printSubgraphList();
}
});

JPanel subgraphP = new JPanel();
subgraphP.setBorder(new LineBorder(Color.decode("#808B96")));
subgraphP.setBounds(578, 170, 296, 174);
frmNetExtranalysis.getContentPane().add(subgraphP);
subgraphP.setLayout(null);

JPanel graphInfoLblP = new JPanel();
graphInfoLblP.setBounds(0, 0, 296, 23);
graphInfoLblP.setBackground(Color.decode("#808B96"));
subgraphP.add(graphInfoLblP);
graphInfoLblP.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));

JLabel subgraphInfoL = new JLabel("Subgraph Elements");
subgraphInfoL.setFont(new Font("Tahoma", Font.BOLD, 12));
graphInfoLblP.add(subgraphInfoL);

subgraphInfoContentP = new JPanel();
subgraphInfoContentP.setBorder(null);
subgraphInfoContentP.setLayout(new BoxLayout(subgraphInfoContentP, BoxLayout.Y_AXIS));
JScrollPane subgraphScroll = new JScrollPane(subgraphInfoContentP);
subgraphScroll.setBounds(0, 23, 296, 151);
subgraphP.add(subgraphScroll);

graphCentralityP = new JPanel();
graphCentralityP.setBorder(new LineBorder(Color.decode("#808B96")));
graphCentralityP.setBounds(578, 385, 296, 144);
frmNetExtranalysis.getContentPane().add(graphCentralityP);
graphCentralityP.setLayout(null);

centLblP = new JPanel();
centLblP.setBackground(Color.decode("#808B96"));
centLblP.setBounds(0, 0, 296, 23);
graphCentralityP.add(centLblP);
centLblP.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel centralityL = new JLabel("Centrality Legend");
centralityL.setFont(new Font("Tahoma", Font.BOLD, 12));
centLblP.add(centralityL);

centralityLegendP = new JPanel();
centralityLegendP.setBounds(5, 22, 286, 111);
graphCentralityP.add(centralityLegendP);
centralityLegendP.setLayout(new GridBagLayout());

JButton generateB = new JButton("Generate");
generateB.setFont(new Font("Tahoma", Font.BOLD, 12));
generateB.setBounds(785, 351, 89, 23);
WindowUtils.setButtonStyle(generateB);
generateB.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
```

```
final HopsWindow window = new HopsWindow(
new JGraphXAdapter<WeightedVertex, WeightedEdgeWithType>(se.generateSubgraph(true, addVertices)));
window.setVisible(true);
window.addWindowListener(new WindowAdapter() {
@Override
public void windowClosing(WindowEvent e) {
if(window.hasSubmitted()){
graphP.colorVertices(graph.vertexSet(),UserConstants.DEFAULT_COLOR);
SubgraphWindow subgraphW = new SubgraphWindow(graph);
se.setHops(window.getHops());
subgraphW.generateSubgraph(se, addVertices);
subgraphW.setVisible(true);
graphP.colorVertices(subgraphW.getChosenVertices(),UserConstants.CHOSEN_COLOR);
}
}
});
}
});
frmNetExtranalysis.getContentPane().add(generateB);

JButton setThresholdB = new JButton("Set Threshold");
setThresholdB.setFont(new Font("Tahoma", Font.PLAIN, 10));
setThresholdB.setBounds(667, 351, 108, 23);
WindowUtils.setButtonStyle(setThresholdB);
setThresholdB.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
resetCentralityP();
double threshold = 1;
if(se != null){
threshold = se.getEdgeThreshold();
}
final ThresholdWindow window = new ThresholdWindow((int)WindowUtils.getHeaviestEdgeWeight(graph), (int) threshold);
window.setVisible(true);
window.addWindowListener(new WindowAdapter() {
@Override
public void windowClosing(WindowEvent e) {
if(window.hasSubmitted()){
int threshold = window.getThreshold();
VertexPicker vp = new VertexPicker(graph);
Set<WeightedVertex> chosenVertices = vp.getChosenVertices(threshold);
graphP.colorVertices(graph.vertexSet(),UserConstants.DEFAULT_COLOR);
graphP.colorVertices(chosenVertices,UserConstants.CHOSEN_COLOR);
graphP.colorVertices(addVertices,UserConstants.CHOSEN_COLOR);
se.setThreshold((double) threshold);

printSubgraphList();
}
}
});
}
});
frmNetExtranalysis.getContentPane().add(setThresholdB);

createMenuBar();
}

private void createMenuBar(){
JMenuBar menuBar = new JMenuBar();

JMenu fileMenu = new JMenu("File");
createFileMenu(fileMenu);

JMenu subgraphMenu = new JMenu("Graph");
createSubgraphMenu(subgraphMenu);

JMenu centralityMenu = new JMenu("Centrality");
createCentralityMenu(centralityMenu);

menuBar.add(fileMenu);
menuBar.add(subgraphMenu);
menuBar.add(centralityMenu);
frmNetExtranalysis.setJMenuBar(menuBar);

}

private void createCentralityMenu(JMenu centralityMenu) {
JMenu degreeMenu = new JMenu("Degree Centrality");
JMenuItem inDegMi = new JMenuItem("In-Degree Centrality");
inDegMi.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
DegreeCentralityComputer dgc = new DegreeCentralityComputer(graph);
Map<WeightedVertex, Double> inDegree = dgc.getInDegreeList();
colorGraphCentralities(inDegree, "In-Degree Centrality");
refresh();

}
});
JMenuItem outDegMi = new JMenuItem("Out-Degree Centrality");
outDegMi.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
DegreeCentralityComputer dgc = new DegreeCentralityComputer(graph);
Map<WeightedVertex, Double> outDegree = dgc.getOutDegreeList();
```

```
colorGraphCentralities(outDegree, "Out-Degree Centrality");
refresh();

}
});
degreeMenu.add(inDegMi);
degreeMenu.add(outDegMi);
centralityMenu.add(degreeMenu);

JMenuItem closeMi = new JMenuItem("Closeness Centrality");
closeMi.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
ClosenessCentralityComputer ccc = new ClosenessCentralityComputer(graph);
Map<WeightedVertex, Double> closeness = ccc.getClosenessList();
colorGraphCentralities(closeness, "Closeness Centrality");
refresh();

}
});
centralityMenu.add(closeMi);

JMenuItem betMi = new JMenuItem("Betweeness Centrality");
betMi.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
BetweenessCentralityComputer bcc = new BetweenessCentralityComputer(graph);
Map<WeightedVertex, Double> betweeness = bcc.getBetweenessList();
colorGraphCentralities(betweeness, "Betweeness Centrality");
refresh();

}
});
centralityMenu.add(betMi);

JMenuItem eigenMi = new JMenuItem("Eigenvector Centrality");
eigenMi.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
EigenvectorCentralityComputer bcc = new EigenvectorCentralityComputer(graph);
Map<WeightedVertex, Double> eigenV;
try {
eigenV = bcc.getEigenvectorList();
colorGraphCentralities(eigenV, "Eigenvector Centrality");
refresh();
} catch (NotConvergedException e) {
JOptionPane.showMessageDialog(frmNetExtranalysis, "The Graph does not converge.");
}

}
});
centralityMenu.add(eigenMi);
}

private void createSubgraphMenu(JMenu subgraphMenu) {
JMenuItem resetMi = new JMenuItem("Reset Graph");
resetMi.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
reset();
}
});
subgraphMenu.add(resetMi);
}

private void reset(){
if (graphP != null)
graphP.colorVertices(graph.vertexSet(),UserConstants.DEFAULT_COLOR);
if(se != null)
se.clearSettings();
addVertices.clear();

resetCentralityP();
elementInfoContentP.removeAll();
elementInfoContentP.revalidate();
elementInfoContentP.repaint();
subgraphInfoContentP.removeAll();
subgraphInfoContentP.revalidate();
subgraphInfoContentP.repaint();
}

private void resetCentralityP(){
centLblP.removeAll();
JLabel lblCentrality = new JLabel("Centrality Legend");
lblCentrality.setFont(new Font("Tahoma", Font.BOLD, 12));
centLblP.add(lblCentrality);
centLblP.revalidate();
centLblP.repaint();
centralityLegendP.removeAll();
centralityLegendP.revalidate();
centralityLegendP.repaint();
}

private void createFileMenu(JMenu fileMenu){
JMenuItem importMi = new JMenuItem("Import graph...");
importMi.addActionListener(new ActionListener() {
```

```
public void actionPerformed(ActionEvent ev) {
FileDialog fd = new FileDialog(frmNetExtranalysis, "Load graph from file...", FileDialog.LOAD);
fd.setVisible(true);

String fn = fd.getDirectory() + fd.getFile();
GraphFileReader fileR;
if(!fn.equals("nullnull")){
try {
fileR = new GraphFileReader(new FileReader(fn));
graph = fileR.generateGraphFromFile();
se = new SubgraphExtractor(graph);

if(graphP != null)
frmNetExtranalysis.getContentPane().remove(graphP);
graphP = new GraphPanel(graph);
graphP.setGraphListener(elementInfoContentP);
graphP.setBorder(null);
//                                          graphP.setBounds(10, 11, 558, 518);
graphP.setBounds(10, 11, 558, 485);
frmNetExtranalysis.getContentPane().add(graphP);
refresh();
reset();

JButton zoomOutB = new JButton("-");
WindowUtils.setButtonStyle(zoomOutB);
zoomOutB.setBounds(519, 506, 49, 23);
zoomOutB.addActionListener(new ActionListener(){
@Override
public void actionPerformed(ActionEvent e) {
graphP.zoomGraph(false);
}

});
frmNetExtranalysis.getContentPane().add(zoomOutB);

JButton zoomInB = new JButton("+");
WindowUtils.setButtonStyle(zoomInB);
zoomInB.addActionListener(new ActionListener(){
@Override
public void actionPerformed(ActionEvent e) {
graphP.zoomGraph(true);
}

});
zoomInB.setBounds(460, 506, 49, 23);
frmNetExtranalysis.getContentPane().add(zoomInB);

} catch (FileNotFoundException e) {
JOptionPane.showMessageDialog(frmNetExtranalysis, "File not Found.");
} catch (IOException e) {
JOptionPane.showMessageDialog(frmNetExtranalysis, "Error Reading the File.");
} catch (Exception e) {
JOptionPane.showMessageDialog(frmNetExtranalysis, "Please check the format of contents of the File.");
}
}
}
});
JMenuItem exportMi = new JMenuItem("Export graph as image...");
exportMi.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
FileDialog fd = new FileDialog(frmNetExtranalysis, "Save graph as image...", FileDialog.SAVE);
fd.setVisible(true);

String fn = fd.getDirectory() + fd.getFile();
try {
if(graphP != null){
BufferedImage image = mxCellRenderer.createBufferedImage(graphP.getGraphAdapter(), null, 1, Color.WHITE, true, null
ImageIO.write(image, "PNG", new File(fn));
}
} catch (IOException e) {
JOptionPane.showMessageDialog(frmNetExtranalysis, "Error Writing the File.");
}

}
});
JMenuItem exitMi = new JMenuItem("Exit");
exitMi.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent ev) {
System.exit(0);
}
});
fileMenu.add(importMi);
fileMenu.add(exportMi);
fileMenu.addSeparator();
fileMenu.add(exitMi);
}

private void refresh(){
frmNetExtranalysis.invalidate();
frmNetExtranalysis.validate();
frmNetExtranalysis.repaint();
}
```

```
private void colorGraphCentralities(Map<WeightedVertex, Double> centralityTable, String centType){
List<WeightedVertex> vList = new ArrayList<>(centralityTable.keySet());
double min = centralityTable.get(vList.get(0)),
max = centralityTable.get(vList.get(vList.size()-1));
double interval = (max - min) / UserConstants.GRADIENT.length;
double curMaxBounds = 0, curMinBounds = 0;
int curVertex = 0;
for(int i = 0; i < UserConstants.GRADIENT.length; i++){
curMaxBounds = min + (i+1)*interval;
curMinBounds = min + i*interval;
Set<WeightedVertex> chosen = new HashSet<>();
while(isInBounds(vList, curVertex) &&
(centralityOfVertex(centralityTable, vList, curVertex) <= curMaxBounds &&
centralityOfVertex(centralityTable, vList, curVertex) >= curMinBounds)){
chosen.add(vList.get(curVertex++));
}
graphP.colorVertices(chosen, UserConstants.GRADIENT[i]);
chosen.clear();
}
setCentralityLegend(centralityTable, centType, min, interval);
}

private Double centralityOfVertex(
Map<WeightedVertex, Double> centralityTable,
List<WeightedVertex> vList, int curVertex) {
return centralityTable.get(vList.get(curVertex));
}

private boolean isInBounds(List<WeightedVertex> vList, int curVertex) {
return curVertex >= 0 && curVertex < vList.size();
}

private void setCentralityLegend(Map<WeightedVertex, Double> centralityTable, String centType, double min, double i
centLblP.removeAll();
JLabel lblCentrality = new JLabel(centType + " Legend");
lblCentrality.setFont(new Font("Tahoma", Font.BOLD, 12));
centLblP.add(lblCentrality);
centLblP.revalidate();
centLblP.repaint();

centralityLegendP.removeAll();
double lowerBound = min, upperBound = 0;
GridBagConstraints gbc = new GridBagConstraints();
gbc.gridwidth = GridBagConstraints.REMAINDER;
for(int i = 0; i < UserConstants.GRADIENT.length; i++){
upperBound = min + (i+1)*interval;
final Color curColor = Color.decode(UserConstants.GRADIENT[i]);

JPanel legendEntryP = new JPanel();
@SuppressWarnings("serial")
JPanel colorBox = new JPanel() {
public void paint(Graphics g) {
super.paint(g);
g.setColor(curColor);
g.fillRect(0, 0, getWidth(), getHeight());
}
};
legendEntryP.add(colorBox);
JLabel limitsL = new JLabel(String.format("%.3f", lowerBound) + " - " + String.format("%.3f", upperBound));
legendEntryP.add(limitsL);
centralityLegendP.add(legendEntryP, gbc);

lowerBound = upperBound;
}
centralityLegendP.revalidate();
centralityLegendP.repaint();
}

private void printSubgraphList(){
Set<WeightedVertex> completeVertexList = new HashSet<>();
completeVertexList.addAll(addVertices);
if(se != null){
VertexPicker vp = new VertexPicker(graph);
completeVertexList.addAll(vp.getChosenVertices(se.getEdgeThreshold()));
}

subgraphInfoContentP.removeAll();
for(WeightedVertex wv : completeVertexList)
subgraphInfoContentP.add(new JLabel(wv.getName()));
subgraphInfoContentP.revalidate();
subgraphInfoContentP.repaint();
}
}
```

HopsWindow.java

```
package window.views;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;
```

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;

import org.jgrapht.ext.JGraphXAdapter;

import com.mxgraph.layout.mxCircleLayout;
import com.mxgraph.layout.mxParallelEdgeLayout;
import com.mxgraph.swing.mxGraphComponent;

import java.awt.Font;

@SuppressWarnings("serial")
class HopsWindow extends JFrame{
JSlider thresholdSlider;
JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter;
boolean hasSubmitted = false;
int hops;

public HopsWindow(JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter) {
this.graphAdapter = graphAdapter;
initialize();
}

private void initialize() {
setTitle("Generate Subgraph...");
setResizable(false);
setBounds(100, 100, 477, 592);
getContentPane().setLayout(null);
setLocationRelativeTo(null);

JPanel subgraphP = new JPanel();
subgraphP.setBounds(10, 11, 437, 439);
subgraphP.setLayout(new BorderLayout());
WindowUtils.setupGraph(graphAdapter);
mxCircleLayout organicLayout = new mxCircleLayout(graphAdapter);
mxParallelEdgeLayout parallelLayout = new mxParallelEdgeLayout(graphAdapter);
organicLayout.execute(graphAdapter.getDefaultParent());
parallelLayout.execute(graphAdapter.getDefaultParent());

mxGraphComponent graphCom = new mxGraphComponent(graphAdapter);
graphCom.setConnectable(false);
subgraphP.add(graphCom, BorderLayout.CENTER);
getContentPane().add(subgraphP);

final JSlider hopSl = createHopsSlider();
hopSl.setBounds(104, 491, 257, 51);
getContentPane().add(hopSl);

JLabel lblNumberOfHops = new JLabel("Number of Hops");
lblNumberOfHops.setBounds(104, 460, 98, 14);
getContentPane().add(lblNumberOfHops);

JButton setValuesB = new JButton("Generate");
setValuesB.setFont(new Font("Verdana", Font.PLAIN, 9));
setValuesB.addActionListener(new ActionListener(){
@Override
public void actionPerformed(ActionEvent e) {
hops = hopSl.getValue();
hasSubmitted = true;
dispatchEvent(new WindowEvent(HopsWindow.this, WindowEvent.WINDOW_CLOSING));
}

});
setValuesB.setBounds(254, 457, 107, 23);
WindowUtils.setButtonStyle(setValuesB);
getContentPane().add(setValuesB);
}

private JSlider createHopsSlider(){
JSlider slider = new JSlider(JSlider.HORIZONTAL, 1, 3, 1);

slider.setMinorTickSpacing(1);
slider.setPaintTicks(true);
slider.setPaintLabels(true);

slider.setLabelTable(slider.createStandardLabels(1));
return slider;
}

public boolean hasSubmitted(){
return hasSubmitted;
}

public int getHops(){
```

```
return hops;
}
}
```

WindowUtils.java

```
package window.views;

import graph.WeightedEdgeWithType;
import graph.WeightedVertex;

import java.awt.Color;
import java.util.Set;

import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import org.jgrapht.ext.JGraphXAdapter;
import org.jgrapht.graph.DirectedWeightedMultigraph;

/**
*
* @author Danielle Rodriguez
*
*/
public class WindowUtils {
static void setupGraph(JGraphXAdapter<WeightedVertex, WeightedEdgeWithType> graphAdapter){
graphAdapter.setCellsEditable(false);
graphAdapter.setCellsResizable(false);
graphAdapter.setAllowDanglingEdges(false);
graphAdapter.setAllowLoops(false);
graphAdapter.setCellsDeletable(false);
graphAdapter.setCellsCloneable(false);
graphAdapter.setCellsDisconnectable(false);
graphAdapter.setDropEnabled(false);
graphAdapter.setSplitEnabled(false);
graphAdapter.setCellsBendable(false);
graphAdapter.setEdgeLabelsMovable(false);
}

static double getHeaviestEdgeWeight(DirectedWeightedMultigraph<WeightedVertex, WeightedEdgeWithType> graph){
double heaviestWeight = 0.0;
Set<WeightedEdgeWithType> edgeSet = graph.edgeSet();
for(WeightedEdgeWithType edge : edgeSet){
double curWeight = graph.getEdgeWeight(edge);
if(curWeight > heaviestWeight)
heaviestWeight = curWeight;
}
return heaviestWeight;
}

static void setButtonStyle(final JButton button){
button.setBorderPainted(false);
button.setFocusPainted(false);

button.setContentAreaFilled(false);
button.setOpaque(true);

button.setBackground(Color.decode("#AEB6BF"));
button.setForeground(Color.decode("#616A6B"));

button.addChangeListener(new ChangeListener() {
@Override
public void stateChanged(ChangeEvent evt) {
if (button.getModel().isPressed()) {
button.setBackground(Color.decode("#2C3E50"));
button.setForeground(Color.decode("#F2F3F4"));
} else if (button.getModel().isRollover()) {
button.setBackground(Color.decode("#808B96"));
button.setForeground(Color.decode("#FBEEE6"));
} else {
button.setBackground(Color.decode("#AEB6BF"));
button.setForeground(Color.decode("#616A6B"));
}
}
});
}

static void setHeaderPanels(JPanel panel){

}
}
```

UserConstants.java

```
package window.views.utils;

public class UserConstants {
public final static String[] GRADIENT = {
"#229954",
"#5A7C49",
```

```
"#925F3F",
"#CB4335"
};

public static String DEFAULT_COLOR = "#ADC5FF";
public static String CHOSEN_COLOR = "#EC7063";
public static String INDIRECT_COLOR = "#FFC800";
public static String SUBGRAPH_BACKGROUND = "#F0F0F0";
}
```

XI. Acknowledgement

This has finally come to an end. Ten months of hard work all culminated in this one document. However, the journey from the start to the finish has not been a solo one. This wouldn't have been completed without the help of various people who have accompanied me in this trip and helped me get to this point.

First off, I'd like to thank my advisers. I thank you Sir Solano for giving me this interesting topic and getting on board with this project where I have learned much, not only about programming but also in other disciplines. The fact that you took the time to teach me and provide various insights while you were still my adviser is something I'm extremely grateful for. I'm also thankful for being patient with me when I've made various mistakes while creating this thing. You're still one of the teachers I really look up to even though I've only been in one of your classes hehe. Speaking of teachers I look up to, I'd also like to thank my "ninong" adviser Sir Ignacio for taking me under his wing when Sir Solano had to leave the country. Even though you had a lot of student under your wing, I'm still grateful that you still let me consult you on occasion regarding my project, even if it isn't your specialty. Thank you again for being helpful and providing very valuable insight.

I would next like to thank my supportive family. To my helpful and supportive sisters, I'm grateful for all your support and telling me to work when I have to. I don't think I'd be able to get this far without your help and your nudging. I would especially like to thank my wonderful, wonderful parents. Not only for providing the funds that I can use when I go the cafe where I can work on this thing, but also for all the support you've given me. For all those times when you've changed the schedule just so that I could work on my project and for all the words of encouragement, I thank you for each and every one of them. I love you Mama and Papa, this is all for you.

To my dear friends, I thank you too for all the support that you've shown and for

believing in me. I'm grateful that you've been the people I've hung out with for the past four years even though we may not have that much in common. Your support means a lot to me and I thank you again for it.

I'd also like to extend my thanks to the teachers that I've encountered, be it from a major subject or not, for teaching me new things that I may or may not use out in the real world. Every new thing I've learned is also a new thing that I've discovered about myself or the world around me. I also thank you Ate Eden for reminding me about the schedule and the required things and also letting me stay in the RH114 if the situation permits it.

And to my favorite birb whose weirdness goes well with mine, I'm grateful for your presence and for your support. You're a wonderful person who I'm thankful to have. Getting close with you is arguably one of the best things to have happened to me. Uvoli.