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

Pathway based Human Disease Clustering and Similarity Analysis Tool Using Frequent Structure Mining

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

Delwyn P. Mendoza

May 2018

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 "Pathway based Human Disease Clustering and Similarity Analysis Tool Using Frequent Structure Mining" prepared and submitted by Delwyn P. Mendoza in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

____

Geoffrey A. Solano, Ph.D. (*cand.*)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*cand.*)	__________	__________
2. Avegail D. Carpio, M.Sc.	__________	__________
3. Richard Bryann L. Chua, Ph.D. (*cand.*)	__________	__________
4. Perlita E. Gasmen, M.Sc. (*cand.*)	__________	__________
5. Marvin John C. Ignacio, M.Sc. (*cand.*)	__________	__________
6. Ma. Sheila A. Magboo, M.Sc.	__________	__________
7. Vincent Peter C. Magboo, M.D., 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

Methods in establishing and understanding human disease similarity are in continuous development as the result from these methods may provide new insights in the field of medicine. Furthermore being able to mine and visualize frequent subgraphs enables the users to view the shared components and relations among the specified diseases. Through the use of a graph mining algorithm called FP-GraphMiner and the pathway database of Kyoto Encyclopedia of Genes and Genomes, graph representation and frequent subgraph mining on human diseases is now possible. Disease Similarity Analyzer is a tool which aims to show disease similarity using hierarchical clustering and visualize frequent substructures in human disease pathways using FP-GraphMiner algorithm.

Keywords: Disease Pathways, Disease Similarity, frequent subgraphs, FP-GraphMiner, Hierarchical Clustering

Contents

List of Figures

I. Introduction

A. Background of the Study

During the past few years, it is evident that methods and tools regarding human disease similarity is continuously being developed. This is because understanding the similarity among various human diseases may provide new information about disease taxonomy, causes and manifestation[1].

Biological pathways are series of intracellular actions that leads to a specific functionality such as the production of a certain product or a change within the cell [2]. Analysis of these pathways that exist in human diseases can show important biological information that could help in disease and medical research. In particular, disease pathways, which are pathways that represent the interactions among the important components of a certain disease such as genes, regulators and metabolites, may provide new information about a certain disease[3].

Pathways are usually the product of manually organizing the information that came from multiple experiments[3]. Because of technological advancements in methods of experimentation in biology, there has been a significant growth of the generated data that can be used in the generation of biological pathways[4]. The generated biological pathways from the experiments have complex forms, however we can represent them using graphs with the nodes as genes, proteins, metabolites, small molecules or chemical compounds and edges as the representation of several biological reactions that occur between them[3]. Using network analysis, we can now view biological pathways at a higher level to have a better understanding of the relationships between its components[4].

Representing biological networks as graphs allows multiple approaches in investigating disease behaviour and similarity. This type of representation allows the viewing of diseases' structure and component relationships as well as the application of graph algorithms that can help in disease comparison and analysis. However only a few of the disease similarity studies convert the biological pathways or reactions to graphs and use them as basis of analysis[5, 6].

There are many databases that contain disease pathway data and one of them is the Kyoto Encyclopedia of Genes and Genomes(KEGG). KEGG is a database resource developed by Kanehisa Laboratories that contains molecular and genomic level information from different organisms. Within KEGG, there is KEGG Disease database containing disease pathways which represents the knowledge of the disease's genetic pathway that is publicly accessible[7].

Due to the recent spark of interest in applying frequent subgraph mining algorithms in various fields such as network traffic analysis[8] or software fault localization[9] to understand and analyze the topology of the network, it is now an important subfield of graph and data mining. This approach can be extended to solve other problems in fields like biology, particularly in mining frequent structures from biological networks.

Among all of the recently developed frequent structure mining algorithms, the time and space efficient FP-GraphMiner by Vijayalakshmi et al[8] stands above the rest as it creates a structure that allows for fast frequent structure mining. Another benefit in using FP-GraphMiner is that it only scans the database once, thus, this algorithm can handle large and complex networks without a significant decrease in performance.

It is shown that similarities of the graph representation of human diseases with respect to their Protein-Protein Interaction(PPI) or symptoms implies similarity of the said diseases. This is an evidence that the similarity derived from analysis of graph representation of diseases' biological pathways can be a reliable metric in showing disease relationship[5].

One of the options to show the similarity among the graph representation of the diseases is by using a dendrogram. A dendrogram is a tree structure that shows the similarity among its leaves. To use this visualization technique, one of the two metrics that are currently used in quantifying diseases shall be used; Jaccard Similarity Index for set-wise comparison[10] and Hamming Distance for element-wise comparison[11].

B. Statement of the Problem

Currently, there is no tool that allows researchers effectively extract and visualize frequent structures among diseases using their disease pathways. However, by using graph mining approach, efficient extraction of sub pathways is possible, helping the researchers in human disease investigation and development of cure to the said diseases.

C. Objectives of the Study

This study aims to create a tool that enables the user to view and analyse disease relationships with respect to their biological pathways. The users of this tool, such as biologists is able to do the following:

1. Update new disease kgml file to the database.
    (a) The user is able to manually add a disease entry to the local database by downloading the kgml file from KEGG and .
    (b) The user is able to download the kgml file from the KEGG database directly using the Rest API.
2. Select diseases to be considered in clustering and analysis.
3. Create a Dendrogram that shows the relationships among the user selected diseases using Jaccard Similarity Index for set-wise comparison or Hamming Distance for element-wise comparison.

4. Show the sub pathway/s that exists on at least a specified percentage of the given subset of diseases.
5. Given a set of diseases, enumerate all diseases that contains the user defined reaction or sub pathway.
6. Show the genes, molecules and other entities in the pathways of the selected diseases according to their topological significance in the given disease set using each entity's weighted average as specified in the FP-GraphMiner algorithm.
7. Export the output dendrogram/s, frequent subgraphs, frequent components as pdf file.

The system must be able to show the following:

1. Similarity Matrix based on the selected similarity measure.
2. Frequency Table
3. Frequent Pattern Graph

D. Significance of the Project

The analysis of relationships between human diseases in terms of their pathways can provide new knowledge that could help in various bodies of knowledge.

By applying Jaccard Similarity Index or Hamming Distance on the graph representation of disease pathways, a dendrogram can be generated. The generated dendrogram shows the researchers the degree of similarity among selected diseases based on their disease pathways, giving another perspective in viewing human disease relationship.

Finding similarity among diseases is an important field in drug and treatment development as it is proven that similar diseases are often cured using the same approach[12]. Therefore, providing a way to find or establish similarities among human diseases can be useful in drug and/or treatment formulation.

One of the most common approach in investigating complex biological network is by breaking down the network into manageable parts.This is because analyzing the whole structure is computationally expensive. Therefore developing a tool that would show common structures in the disease set will help system biologists determine what network should be considered in studying human diseases.

E. Scope and Limitations

The goal of this study is to develop a software that enables researchers to view the relative relationships of diseases based on their pathways and perform few analysis on them. The scope and limitations of this study are as follows:

1. The tool only supports .kgml files from KEGG Database.

2. The tool does not show the relationship where the product of a reaction in disease X causes or is a reactant in disease Y.

3. Revisions of the .kgml file must be done only by KEGG.

4. The tool's download pathways files from KEGG functionality displays all the available pathways from the KEGG Pathway Database and its the user's responsibility to download only human disease pathways.

II. Review of Related Literature

One of the greatest challenges in medicine is understanding the relationships among diseases based on their hidden biological mechanisms [6]. To address this challenge, various efforts in research has been made such as studying clinical symptoms, electronic medical records, disease-related genes, micro-RNAs, proteins, pathways and other disease-related reactions [1]. Although there are remarkable advances in the understanding of human diseases, most aspects of phenotype and genotype relationship is still unclear [5].

There are several studies about disease similarity that used certain disease pathways or other phenetic data. DNetDB by Yang et al [1] is an example of a study that tackled the disease similarity problem using the diseases' gene expression data. By using differential coexpression analysis, DNetDB is able to show the common dysfunctional regulation mechanism shared between two diseases by allowing its users to use their 1,326 inferred disease relationships from differential gene coexpression feature similarity.

Le et al[13] also conducted a study on human disease similarity but instead of using gene expressions, they used ontology to derive the disease similarity. In this study, they calculated the similarity for a set of 9,221,365 disease phenotype using ontology-based semantic measures and assessed their results by correlating their results with the disease similarity database from Online Mendelian Inheritance in Man(OMIM) records. They also compared human ontology phenotype to a large number of semantic similarity measures to show that the output works together with gene ontology.

The usage of disease pathways as an approach to functionally characterize diseases was enforced by the study conducted by Li and Agarwal[14]. Using Medical Subject Headings(MeSH) terms and MEDLINE abstracts, they mapped diseases to genes, then associated disease related genes to biological pathways. According to the study, the number of genes from each disease statistically associated with pathways is highly significant ($P - value < 0.001$).

Sun et al[6] used Protein-protein interactions(PP1), Gene Ontology(GO) annotations and disease-gene associations to propose three disease similarity measures. This study modelled PPIs and disease association as graphs which were then analyzed to formulate disease similarity annotation(GO) measure, functional similarity based measure(disease-gene association) and topology-based measure(PPI).

Another study conducted by Sarmiento et al[2] also looked into human disease similarity used shared disease pathways. They developed a tool called PathSOM which uses the disease pathways data from KEGG in kgml format to visualize the similarity among diseases. This tool uses a specific type of artificial neural network called Self-Organizing Maps(SOM) to calculate the derived similarity of a specified group of diseases. By using the calculated similarities from SOM, the tool then allows its users to visualize the output using Lattice visualization, U-Matrix visualization and cluster them using K-means clustering and hierarchical clustering.

Human Symptom-Disease Network (HSDN) by Zhou et al [5] is a symptom-based disease similarity network that is capable of showing disease relationship based on similar genes or protein-protein interactions(PPI). They achieved this by using the 7,109,429 extracted PubMed bibliographic records from the Medical Subject Headings(MeSH) metadata field to get 4,442 disease terms and 322 symptom terms which are then used to derive symptom similarities for each disease pair. The resulting HSDN is a dense graph which shows the similarity between 4,219 diseases connected by 7,488,851 links of positive similarity.

Theodosiou et al[4] developed a web tool named Network Analysis Profiler(NAP) which is mainly designed for network topological analysis and comparison of biological networks. Its users are able to upload their graphs in the form of pairwise connections or create the graph on the application itself. The web tool offers various topological analysis features derived from graph theory which helps in the biological network analysis.

Currently, there are various ways to calculate and visualize similarity among diseases such as Self-Organizing Maps[2] or graphs with edges with weights or thickness directly proportional to the degree of similarity between diseases[5]. Another visualization method that would help in the analysis analysis of disease relationship would be the dendrogram. A dendrogram is a the result given by hierarchical clustering, a method in which the objects in consideration are grouped according to their similarity of dissimilarity[15]. This method of visualization returns a branching diagram that displays inferred relationships among biological entities based on phenetic data[16].

A survey on Hierarchical Clustering methods in Data mining was conducted by Dabhi et al[15] which provided an extensive discussion of the clustering approaches. There are two general approaches for hierarchical clustering, agglomerative and divisive. In the agglomerative or the bottom-up approach, the clustering starts with n leaf nodes and for each step, these nodes are combined to form the clusters. In the divisive or the top-down approach, the clustering starts with one node that is split until it reaches the number of desired. Under agglomerative approach, there are three types of algorithms that determine what node will be merged or separated. The first algorithm is the Single Link Algorithm, also known as nearest-neighbor clustering algorithm which uses the minimum distance to compute the distance between clusters. The second algorithm is the Complete Link Algorithm, also known as farthest-neighbor algorithm where the method uses the maximum distance as a metric for measuring the distance between clusters. The final algorithm is the Average Link Algorithm where the distance between nodes is defined by the average distance between nodes.

There are various metrics that can be used to quantify the distance between objects using categorical data. Osorio et al[10] noted that the Jaccard Similarity Index(JSI) is one of the metrics that can be used to quantify categorical data such as absence or presence of a property. Biological association and clustering studies [10][11] used Hamming Distance as another metric for calculating the similarity/dissimilarity of two sets. Wang et al[11] noted that Hamming Distance is a widely applied metric that retains the component-wise information about the sets being compared.

H. Dinari and H. Naderi[17] conducted a survey of the algorithms that can be used in frequent structure or subgraph mining. They classified each algorithm according to their approach, search strategy, input and output. One of the algorithms that were part of the review is the FP-GraphMiner algorithm which uses FP-growth method to find the frequent subgraphs from the set of graphs in a database. This algorithm assigns a value called BitCode for each edge which denotes the existence of the edge in the set of graphs. The BitCode is then used to construct an FP-tree which is designed to mine the frequent subgraphs using depth traversal.

The study conducted by Osorio et al[10] constructed a phenogram to display the topological similarities of organisms with respect to their metabolic pathways, particularly the glycolysis and citrate cycle. By representing the said pathways as graphs, they applied frequent subgraph mining (FP-GraphMiner Algorithm) to derive the similarities among the set of metabolic pathways from various organisms, allowing them to prove that metabolic pathway network topology is a significant source of data for analyzing similarities of taxa.

The study by Thilaga et al[18] shows the application of subgraph mining in the field of medicine and diseases. Due to the complexity of the brain, particularly its neuronal interactions, the need for computational models to analyse the neural mechanisms within the brain is apparent. In this study, they denoted the neuronal interactions as a functional brain network and applied graph pattern mining to help in the analysis of brain function during cognition and to possibly help in the detection cognitive damages that may cause mental problems.

Thomas et al[3] looked into graph clustering and subgraph similarity on neurological disorders. They showed that applying graph mining approaches biological and brain networks can help in understanding the biological workings behind neurological disorders like Alzheimer's and Parkinson's diseases. Using the said approach in other brain networks such as the structural and functional networks also help in understanding the nature of neurological disorders.

III. Theoretical Framework

A. Kyoto Encyclopedia of Genes and Genomes(KEGG)

Kyoto Encyclopedia of Genes and Genomes(KEGG) is a database resource that contains molecular and genomic level information from different organisms. It consists of eighteen database that are separated by different categories such as systems information, genomic information, chemical information and health information[7]. KEGG also offers some pathway analysis functionalities such as KEGG Mapping for pathway mapping, BlastKOALA for genome annotation and BLAST/FASTA for sequence similarity. However, KEGG doesn't have visualization functionality other than its pathway image[7]. Within KEGG, there is KEGG Disease database containing disease pathways which represents the knowledge of the disease's genetic pathway that is publicly accessible.

The KEGG Markup Language(kgml) is the file format used by KEGG for their pathways. This markup language allows computational analysis and modelling of gene or protein networks. Every pathway in the KEGG Database has its own kgml file representation which is downloadble using the ”Download KGML” functionality located at each pathway page or by using the *get* method of KEGG API[19].

B. Disease Pathways

Disease pathways shows the relationships among the components of a certain disease such as genes, compounds and metabolites. Disease pathways are treated as a collection of related pertubed states of the molecular system. Other factors such as genetic factors, environment and drugs are also considered as disturbances to the molecular system[20]. These disease pathways are used to visualize the underlying disease mechanism or as basis for studying and understanding human disease and their relationships with each other[2, 5, 6].

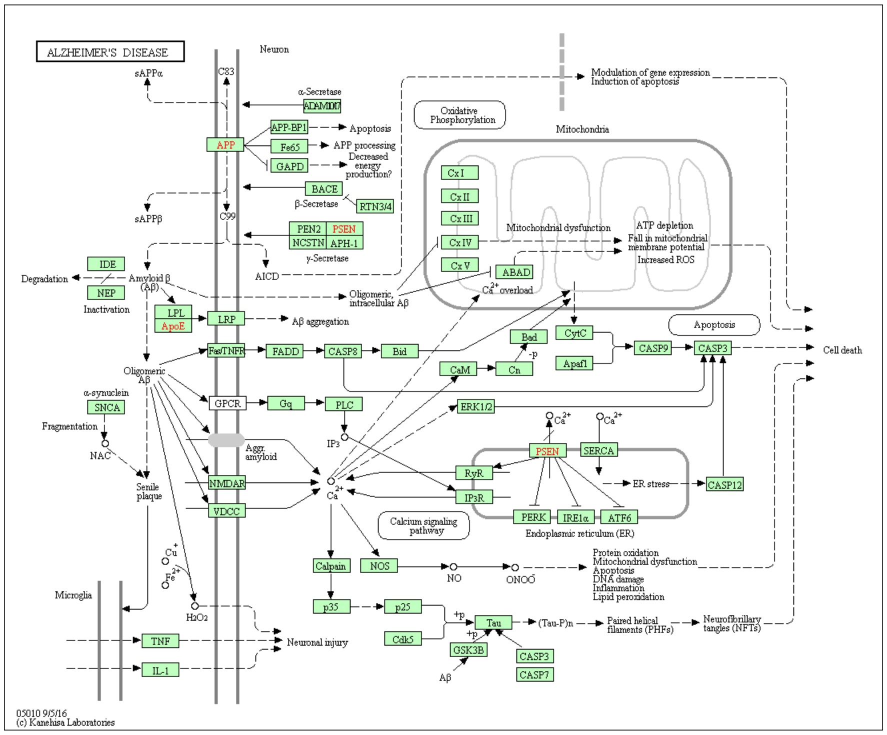


Figure 1: Alzheimer's Disease Pathway

C. Disease Relationship

It is widely accepted that genes within cells do not function on their own. It is shown that these genes interact with each other to create pathways or complexes that do a certain biological functionality. In the case of diseases, these genes can be functionally be related by biological pathways that lead to a certain clinical phenotype. This relationship allows researchers to investigate disease relationship based on their related pathways and genes, hoping to acquire information that would help understanding the diseases and disease relationship[14].

Comprehending disease relationship using their underlying biological mechanisms is one of modern biology's challenges. The current information about disease relationship in aspects such as phenotypes and genotypes, particulary for complex diseases are still unclear[5]. Using the derived relationships among human diseases, it is expected that we would gain new knowledge that would help in disease diagnosis and treatment[6].

D. FP-GraphMiner

FP-GraphMiner is a graph mining algorithm by Vijayalakshmi et al[8] that gets frequent patterns in a graph database. The following are the concepts and methods introduced in the algorithm:

D..1 Graph Database

A database $GD = \{G_1, G_2, ..., G_k\}$ that contains k graphs.

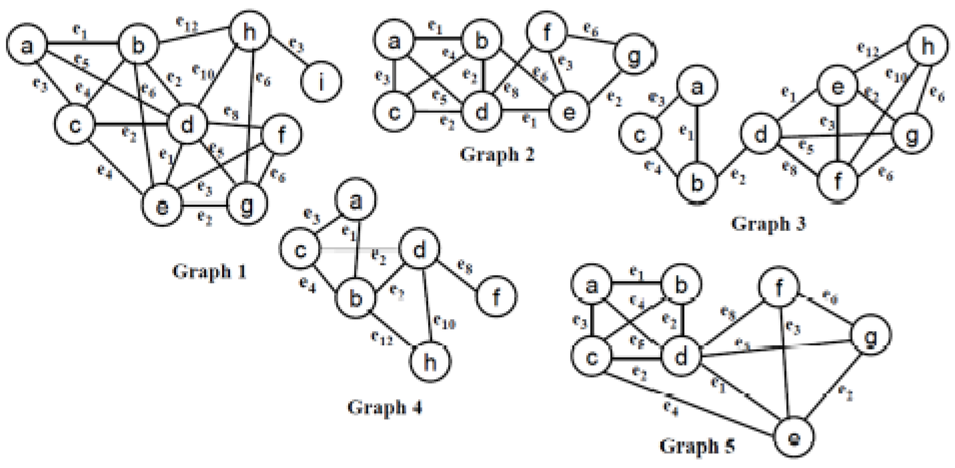


Figure 2: Graph Database

D..2 BitCode of a Distinct Edge

The BitCode $BitCode(DE_i)$ of a distinct edge DE_i is a k-length string, where k is the number of graphs in the graph database GD. The ith character of the $BitCode(DE_i)$ is 1 if it is present on the ith graph in GD and 0 otherwise.

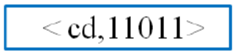


Figure 3: Distinct Edge with its BitCode

D..3 Weight of a BitCode

The weight of a BitCode, $WT(DE_i)$ is the number of 1's in it. This denotes the number of graphs where DE_i exists. For example, based on figure 3, the weight of the distinct edge cd is $WT(cd) = 4$

D..4 Node

A node is a collection of subgraphs with the same BitCode.

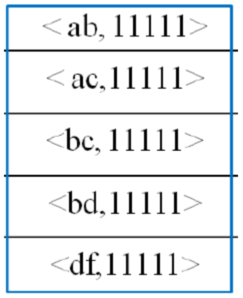


Figure 4: Node with BitCode (11111)

D..5 Cluster

A cluster is a collection of nodes that has the same BitCode weights.

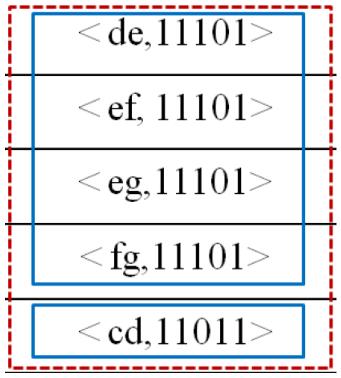


Figure 5: Cluster with BitCode Weight = 4

D..6 Support σ

The support σ for a given frequent pattern or distinct edge is defined as the ratio of the graphs that contain the frequent pattern or distinct edge to the total number of graphs.

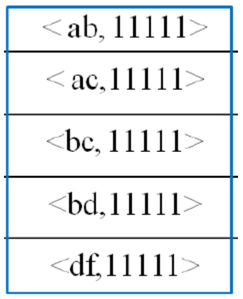


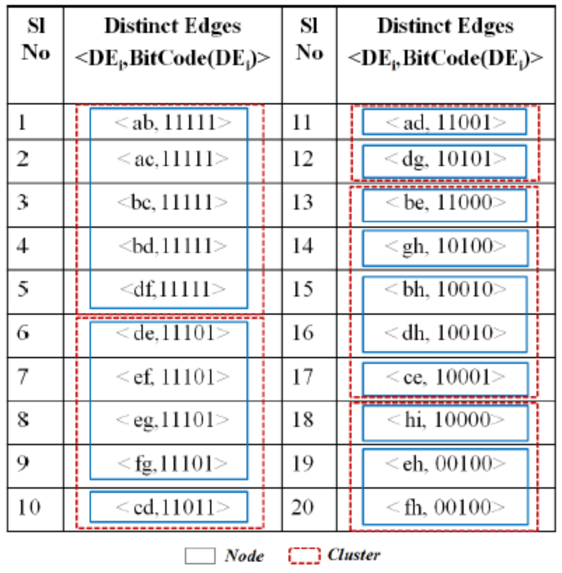
Figure 6: Node with 100% support

Figure 7: Distinct Edge with 80% support

D..7 Frequency Table

The Frequency Table FT is a collection of 2-tuple $DE_i, BitCode(DE_i)$ that is arranged in decreasing order based on the binary encoding of every distinct edge DE_i's

BitCode.



Sl No	Distinct Edges <DE$_i$,BitCode(DE$_i$)>	Sl No	Distinct Edges <DE$_i$,BitCode(DE$_i$)>
1	<ab, 11111>	11	<ad, 11001>
2	<ac,11111>	12	<dg, 10101>
3	<bc, 11111>	13	<be, 11000>
4	<bd,11111>	14	<gh, 10100>
5	<df,11111>	15	<bh, 10010>
6	<de,11101>	16	<dh, 10010>
7	<ef, 11101>	17	<ce, 10001>
8	<eg,11101>	18	<hi, 10000>
9	<fg,11101>	19	<eh, 00100>
10	<cd,11011>	20	<fh, 00100>

Node
Cluster

Figure 8: Example of a Frequency Table

D..8 Frequent Pattern Graph

The Frequent Pattern Graph, $FP - Graph = \{Node, Edge\}$ is a special type of undirected graph where the $Node$ is defined as a collection of distinct edges with a common property and an $Edge$ is a link between two $Nodes$.

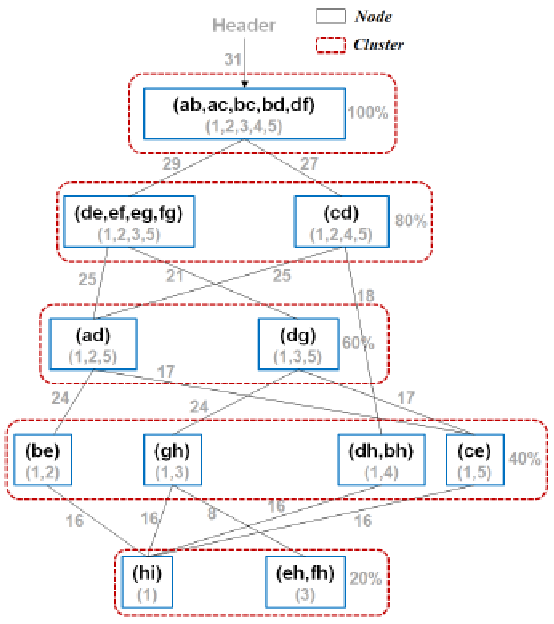


Figure 9: Example of a Frequent Pattern Graph

D..9 Algorithm: FP-Graph Construction

1. For all distinct edge DE_i in the graph database GD, create a 2-tuple:

$$(DE_i, BitCode(DE_i)).$$

2. Create the Frequency Table by sorting the 2-tuples according to their decreasing BitCodes.

3. Form nodes by grouping the 2-tuples with the same BitCodes. Create clusters by grouping all nodes with same BitCode weights.

4. Connect the nodes from different clusters if the distance between the two nodes, defined as $BitCode(p_i) \cap BitCode(p_j)$, is equal to $BitCode(p_i)$ where p_i is from a cluster with lower support than p_j's cluster.

The output of this algorithm, shown in Figure 9 is a special undirected graph that will be used in FP-GraphMiner algorithm.

D..10 Algorithm: FP-GraphMiner

1. Finding all Frequent subgraphs given $\sigma\%$ as support.

   Given the FP-Graph of the graph database GD, acquire all the frequent subgraphs by performing a DFS walk starting from the $Cluster$ with the specified support $\sigma\%$ to the $HeaderNode$

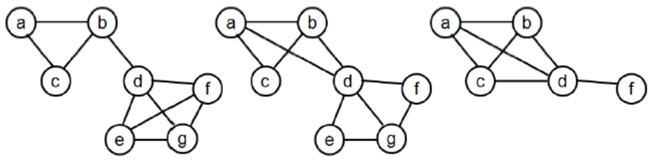


Figure 10: Frequent Subgraphs with 60% support

2. Finding graphs containing an input graph Q

   Given an input graph Q and the FP-Graph, perform a BFS search from the $HeaderNode$ and store every acquired edge's BitCode in an array $BFS(Q)$ until all the edges in Q are obtained. Perform AND operation on the BitCodes in $BFS(Q)$, returning a BitCode that denotes all the graphs that contain the input graph Q.

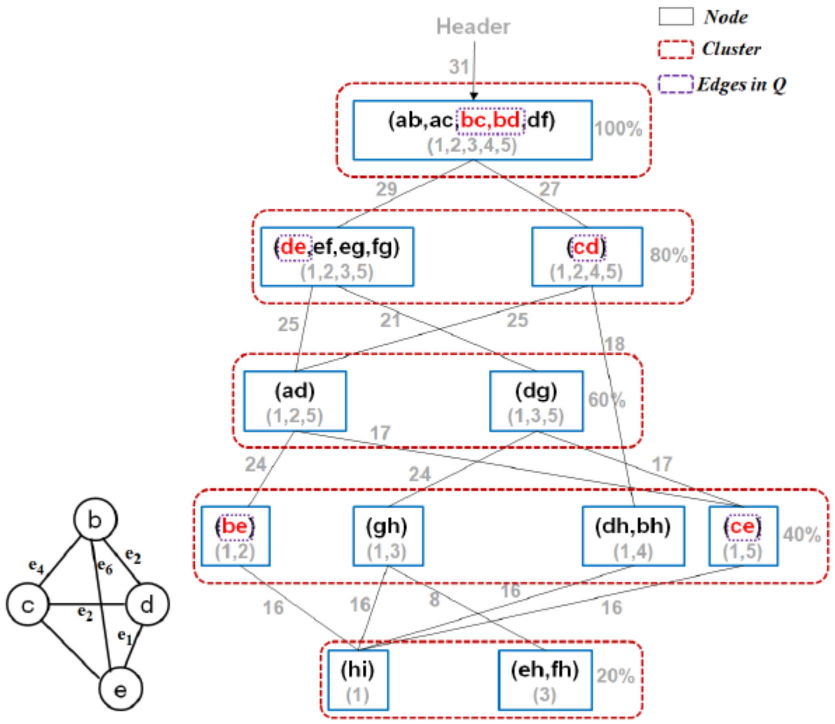


Figure 11: Finding input graph Q

3. Computing the significance of each node in the network.

   Let D be the maximum degree a node in the graph database can have and $\sigma(node_i)$ be the support of the $node_i$'s cluster, the weighted average of each $node_i$ can be computed using the formula:

$$WA_i(node_i) = \lfloor \sum (Degree(node_i) * \sigma(node_i))/D \rfloor, 1 \leq i \leq D$$

   Using figure 2 and figure 9, we have the table of nodes with their support.

σ	a	b	c	d	e	f	g	h	i
100	2	3	2	2	0	1	0	0	0
80	0	0	1	2	3	2	2	0	0
60	1	0	0	2	0	0	1	0	0
40	0	3	1	1	2	0	0	3	0
20	0	0	0	0	1	1	0	3	1

Figure 12: Table of nodes and support values

Based on the figure above, we can compute each of the node's weighted average depending on its support value. For example, node a's weighted average is:

$$WA(a) = \lfloor \tfrac{(2*100)+(1*60)}{8} \rfloor = 32$$

E. Jaccard Similarity Index(JSI)

The JSI of sets A and B is the quotient between their intersection and union.

$$JSI = \frac{A \cap B}{A \cup B}$$

Jaccard Similarity Index can also be used to calculate the distance, known as the Jaccard Distance between two sets A and B using the formula:

$$JSI = 1 - \frac{A \cap B}{A \cup B}$$

Jaccard Similarity Index can be applied to categorical data such as the presence or absence of a property[10]. JSI is also a distance measure still widely used in fields such as biology[21].

F. Hamming Distance

Given two strings of equal length, Hamming distance is the total number of times where the *ith* character from each strings are different. This distance metric is typically used to determine the similarity among discrete data while maintaining the data's component-wise information[11].

G. Hierarchical Clustering

Hierarchical clustering is a method that tries to break down a set of objects into a hierarchy of groups, which is then represented as a tree structure. There are two approaches used in the hierarchical clustering method namely agglomerative and divisive approaches[15].

Between the two approaches in this clustering method, agglomerative or bottom-up approach is more commonly used[2]. In this approach, the number of clusters m is reduced at each iteration.

H. Dendrogram

A type of tree diagram that is formed by hierarchical clustering algorithms and is usually used to cluster data and visualize the similarity among its leaf nodes. Dendrogram is one of the ways to visualize the output of data clustering for exploratory data analysis. This type of visualization is currently being used in practical domains such as image processing, classification, pattern recognition etc[15].

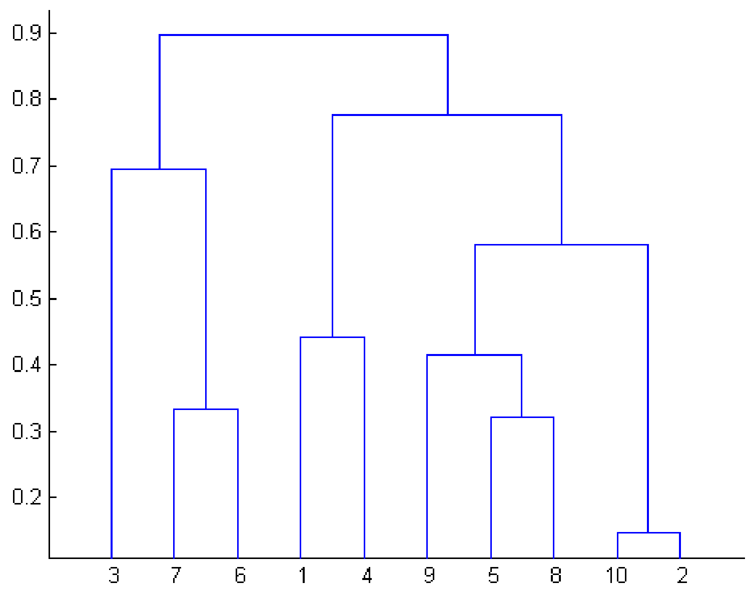


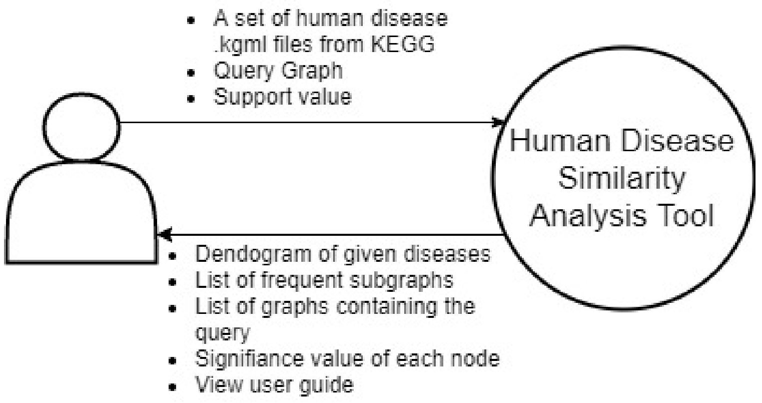
Figure 13: Example of a Dendrogram

The figure above shows a dendrogram where the distance between the two elements is seen by the y value of the point connecting the two elements. Using this dendrogram, we can draw several statements regarding the similarity of its nodes like: the nodes 10 and 2 are the closest nodes.

IV. Design and Implementation

A. System Overview

The implementation of this application is described by the following diagrams. Figure 5 shows the overview of the interaction between the user and the application by defining the user inputs and the expected outputs. The application takes in the untampered kgml files from the user, regardless of the function to be performed. Then the user needs to supply the required data depending on the functionality requested, such as the query graph Q for searching diseases or the support value $\sigma\%$ for searching frequent sub-reactions and so on.



A set of human disease .kgml files from KEGG
Query Graph
Support value
Human Disease Similarity Analysis Tool
Dendogram of given diseases
List of frequent subgraphs
List of graphs containing the query
Signifiance value of each node
View user guide

Figure 14: Input and Output for Human Disease Similarity Analysis Tool

B. Use Case Diagram

Figure 6 defines the actions that can be performed by the users in the application. The figure shows that the user has three main action types. The first is viewing the user guide which would help the user perform the disease analysis and other tasks properly. The next function is adding a disease kgml file from KEGG to the tool's local database. This kgml file must not be tampered or corrupted by any means as it would affect the performance of the tool. The other function, analysis functions, includes several options that revolve around showing the relationship among the user selected diseases.

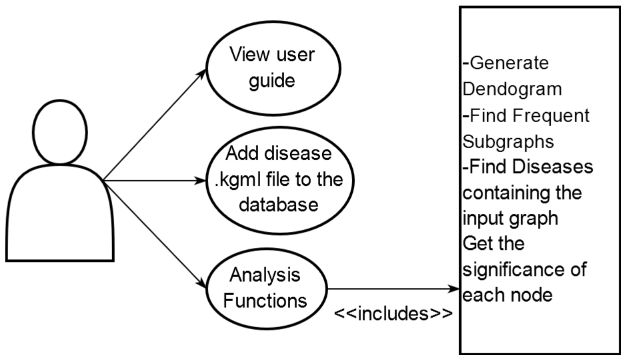

View user guide
Add disease .kgml file to the database
Analysis Functions
<<includes>>
-Generate Dendogram
-Find Frequent Subgraphs
-Find Diseases containing the input graph
Get the significance of each node

Figure 15: Use case Diagram for Human Disease Similarity Analysis Tool

C. Flowchart

To use the application's disease analysis functions, the user must select the human diseases to be studied. The tool will then check if the disease information is stored in the local database, giving the user the opportunity to download the kgml file if necessary. After setting up the disease set, the user will select the action that will be performed by the tool from the main options namely; Finding the frequent sub-reactions, finding diseases with specified pathway and creating the dendrogram. After the analysis, the user is allowed to export the results of the performed functionalities.

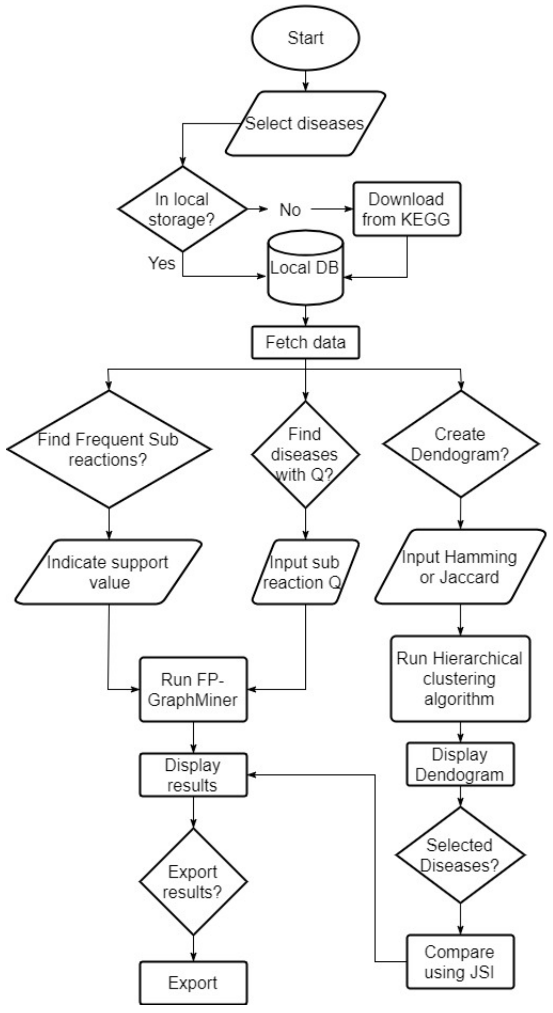


Figure 16: Top level Flowchart for Human Disease Similarity Analysis Tool

The following figure shows the process of creating the Frequent Pattern Graph used by the FP-GraphMiner algorithm to find frequent structures, find diseases with the specified substructure etc.

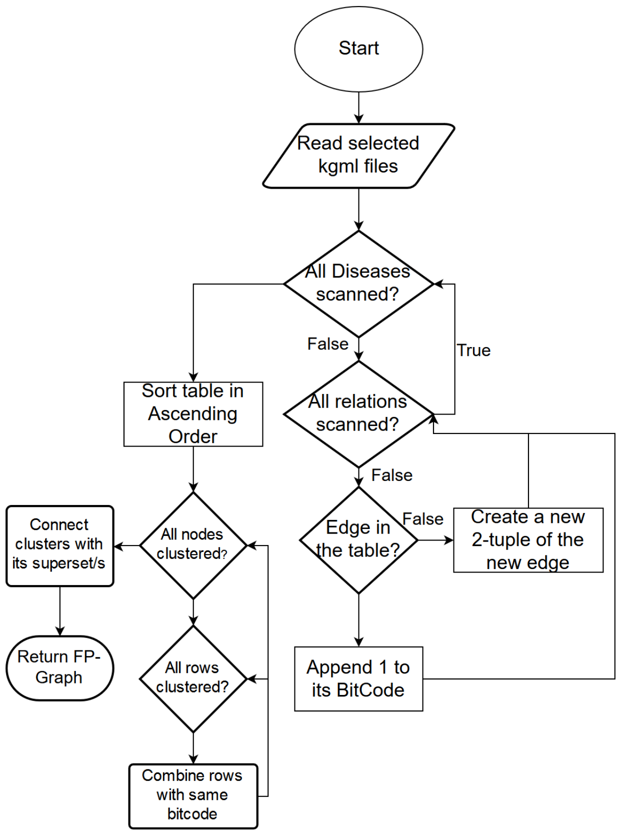


Figure 17: Top level Flowchart for FP-Graph Construction

V. Results

On the start of this tool, Disease Similarity Analyzer will display its main window. This window contains the major options that is currently available to its users, as shown in Figure 18. The user is able to select from the showed options as well as go back to this screen if needed.

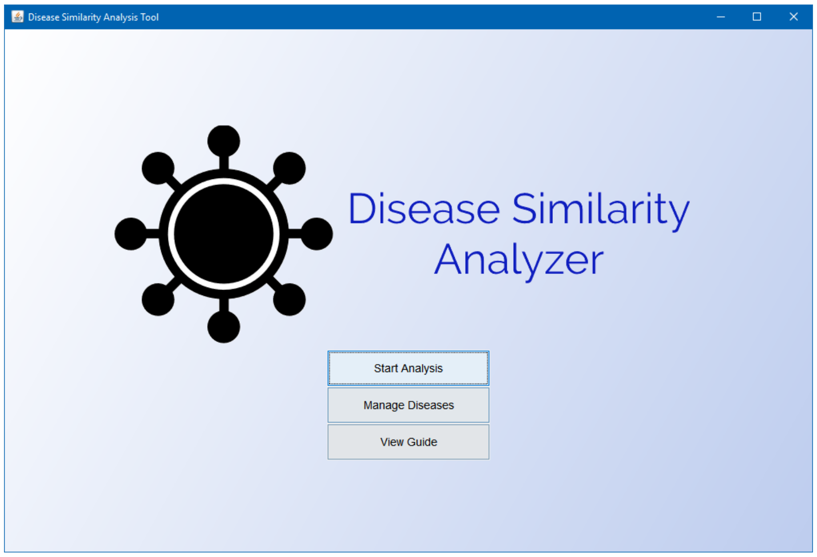


Figure 18: DSA Splash Screen

Once the user selects Start Analysis option, the disease selection panel as shown in Figure 19 will display all the currently available disease pathways for analysis. Left clicking a disease entry in the list will show the disease pathway image from KEGG. The user is required to select at least two disease pathway data before the executing analysis.

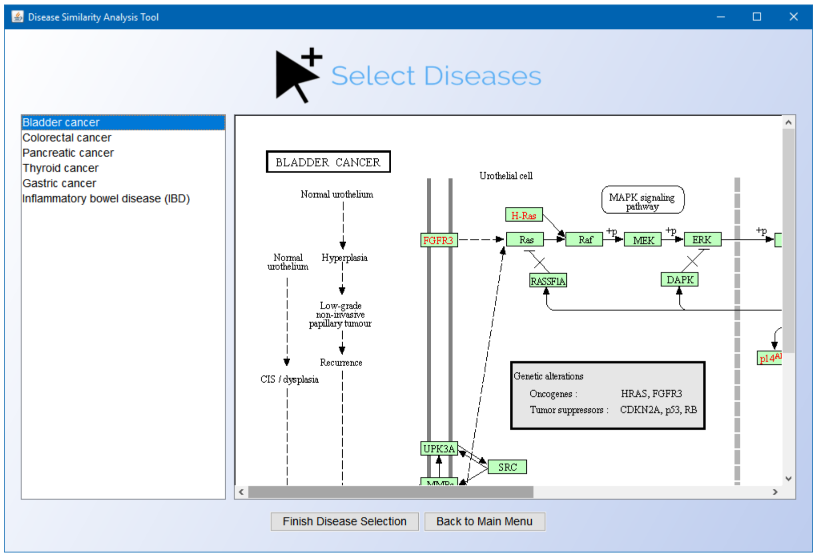


Figure 19: DSA Disease Selection Screen

After selecting the diseases that will be considered in the analysis, the tool will display its functionality menu screen as shown in Figure 20 where the user can select the type of analysis that will be performed on the previously selected disease pathways. The user may hover on the buttons to display a simple tooltip containing a short description of the hovered functionality.

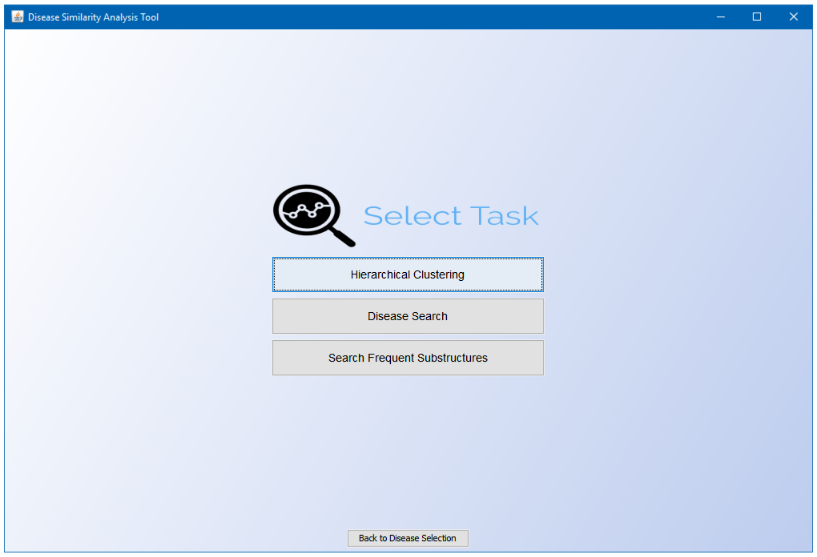
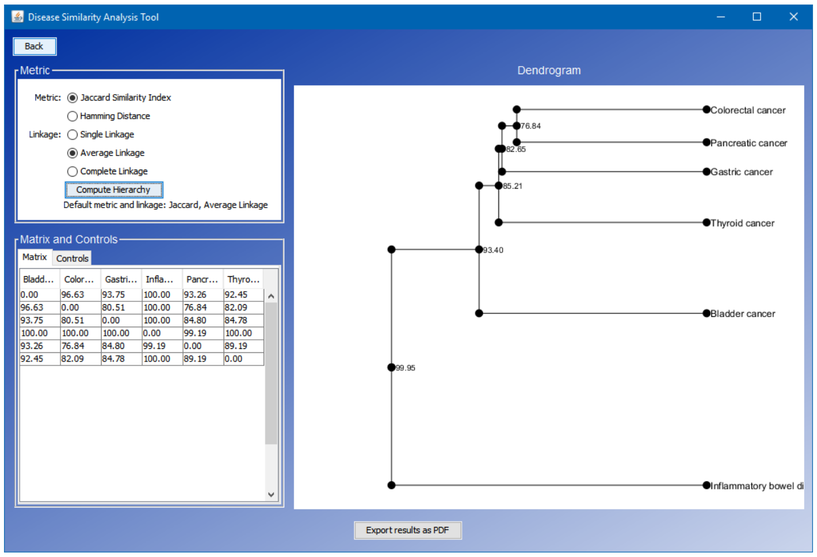

Disease Similarity Analysis Tool
Select Task
Hierarchical Clustering
Disease Search
Search Frequent Substructures
Back to Disease Selection

Figure 20: DSA Functionality Menu Screen

If the user selects Hierarchical clustering from the functionality menu screen, the clustering window, as shown in Figure 21 will be displayed. This window allows the users to select the needed metric type and linkage type for the agglomerative hierarchical clustering. Once the user clicks the Compute Hierarchy button, the output dendrogram will be displayed as well as the computed similarity matrix based on the metric type that was specified by the user.

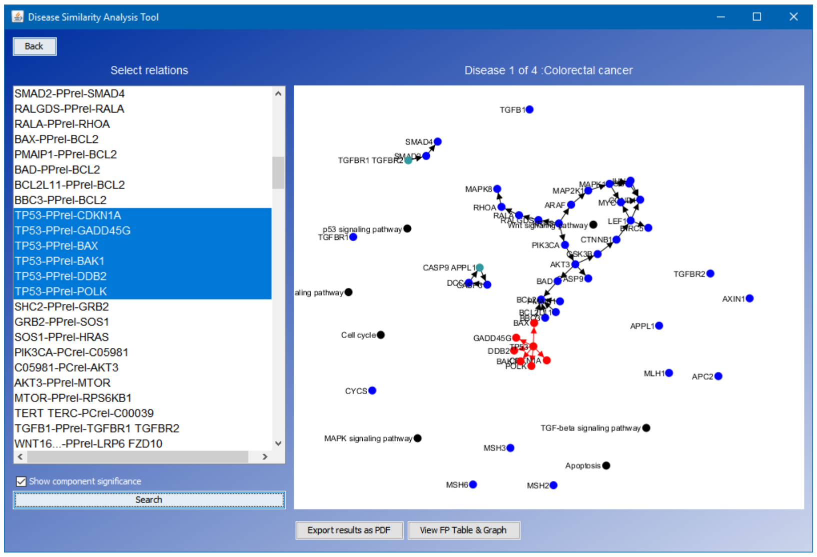


Metric: [✓ Jaccard Similarity Index] Hamming Distance
Linkage: Single Linkage [✓ Average Linkage] Complete Linkage
Default metric and linkage: Jaccard, Average Linkage

Bladd...	Color...	Gastri...	Infla...	Pancr...	Thyro...
0.00	96.63	93.75	100.00	93.26	92.45
96.63	0.00	80.51	100.00	76.84	82.09
93.75	80.51	0.00	100.00	84.80	84.78
100.00	100.00	100.00	0.00	99.19	100.00
93.26	76.84	84.80	99.19	0.00	89.19
92.45	82.09	84.78	100.00	89.19	0.00

Figure 21: DSA Hierarchical Clustering

If the user selects Disease Search in the previous menu, a window as shown in Figure 22 will be display. This window allows the selection of relations which will then be searched across the selected disease pathways.



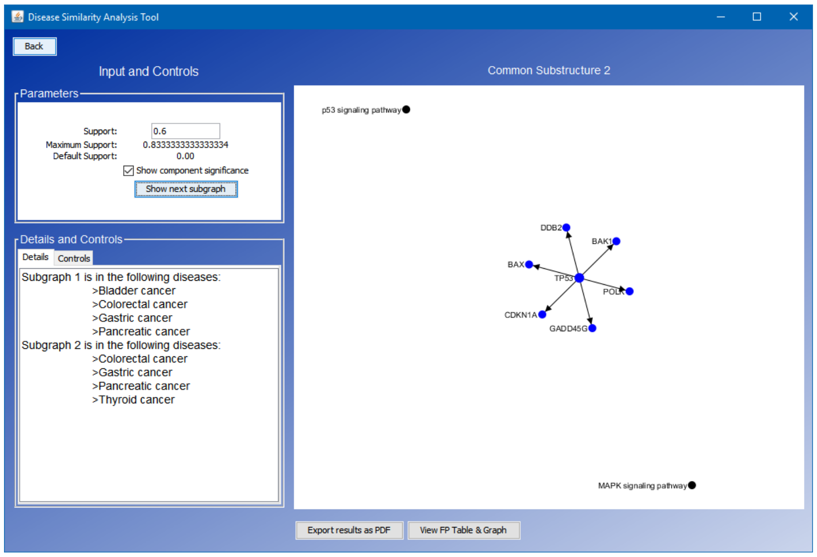
SMAD2-PPrel-SMAD4
RALGDS-PPrel-RALA
RALA-PPrel-RHOA
BAX-PPrel-BCL2
PMAIP1-PPrel-BCL2
BAD-PPrel-BCL2
BCL2L11-PPrel-BCL2
BBC3-PPrel-BCL2
TP53-PPrel-CDKN1A
TP53-PPrel-GADD45G
TP53-PPrel-BAX
TP53-PPrel-BAK1
TP53-PPrel-DDB2
TP53-PPrel-POLK
SHC2-PPrel-GRB2
GRB2-PPrel-SOS1
SOS1-PPrel-HRAS
PIK3CA-PCrel-C05981
C05981-PCrel-AKT3
AKT3-PPrel-MTOR
MTOR-PPrel-RPS6KB1
TERT TERC-PCrel-C00039
TGFB1-PPrel-TGFBR1 TGFBR2
WNT16...-PPrel-LRP6 FZD10

☑ Show component significance

Search

Figure 22: DSA Relation Search

If the user selects the Subgraph Search functionality, the tool will display a window as shown in Figure 23 where the user can search for the frequently occurring relations within the selected disease pathways.



Parameters

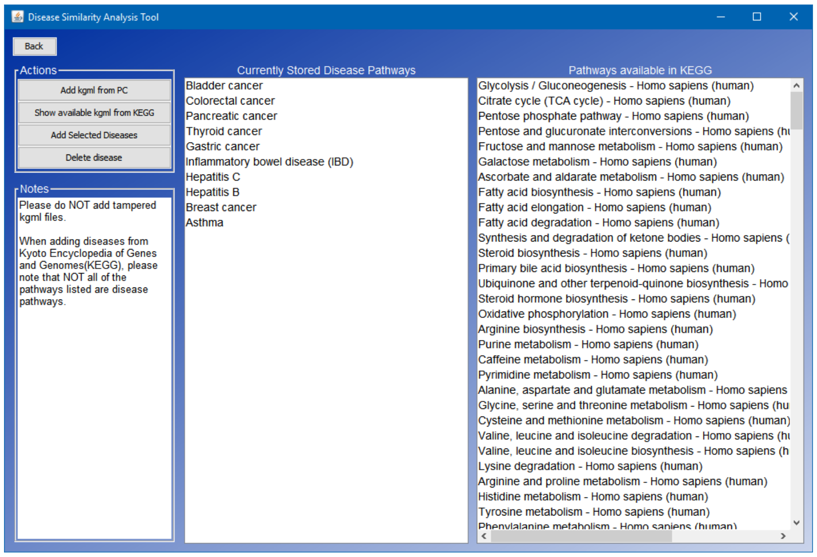
Support: 0.6
Maximum Support: 0.8333333333333334
Default Support: 0.00
☑ Show component significance
Show next subgraph

Details and Controls
Details | Controls

Subgraph 1 is in the following diseases:
>Bladder cancer
>Colorectal cancer
>Gastric cancer
>Pancreatic cancer
Subgraph 2 is in the following diseases:
>Colorectal cancer
>Gastric cancer
>Pancreatic cancer
>Thyroid cancer

Figure 23: DSA Frequent Subgraph Search

In the main menu, there is a Manage Diseases option which directs the user to a panel as shown in Figure 24. This is the window where the user either adds a kgml file from the file system or the KEGG Pathway Database, or delete the selected disease pathway/s from the local database.


Disease Similarity Analysis Tool
Back
Actions
Add kgml from PC
Show available kgml from KEGG
Add Selected Diseases
Delete disease
Notes
Currently Stored Disease Pathways
Pathways available in KEGG

Please do NOT add tampered kgml files.

When adding diseases from Kyoto Encyclopedia of Genes and Genomes(KEGG), please note that NOT all of the pathways listed are disease pathways.

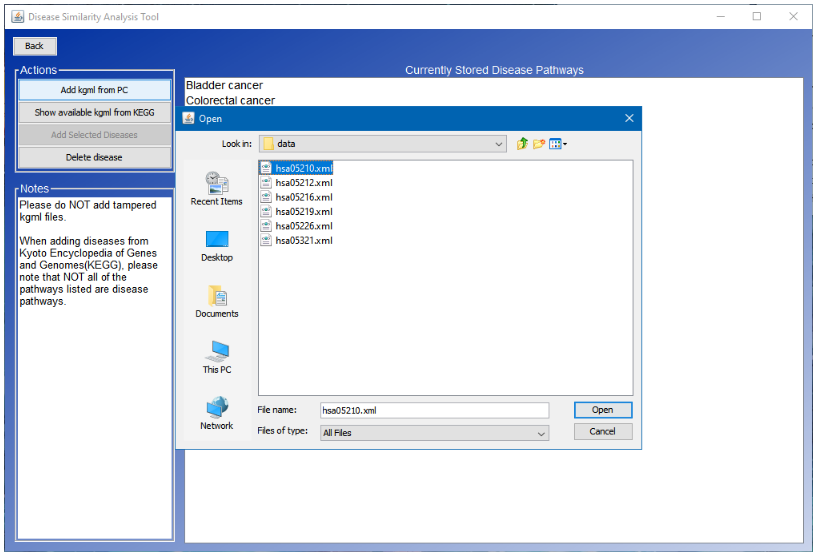
Currently Stored Disease Pathways

- Bladder cancer
- Colorectal cancer
- Pancreatic cancer
- Thyroid cancer
- Gastric cancer
- Inflammatory bowel disease (IBD)
- Hepatitis C
- Hepatitis B
- Breast cancer
- Asthma

Pathways available in KEGG

- Glycolysis / Gluconeogenesis - Homo sapiens (human)
- Citrate cycle (TCA cycle) - Homo sapiens (human)
- Pentose phosphate pathway - Homo sapiens (human)
- Pentose and glucuronate interconversions - Homo sapiens (hu
- Fructose and mannose metabolism - Homo sapiens (human)
- Galactose metabolism - Homo sapiens (human)
- Ascorbate and aldarate metabolism - Homo sapiens (human)
- Fatty acid biosynthesis - Homo sapiens (human)
- Fatty acid elongation - Homo sapiens (human)
- Fatty acid degradation - Homo sapiens (human)
- Synthesis and degradation of ketone bodies - Homo sapiens (
- Steroid biosynthesis - Homo sapiens (human)
- Primary bile acid biosynthesis - Homo sapiens (human)
- Ubiquinone and other terpenoid-quinone biosynthesis - Homo
- Steroid hormone biosynthesis - Homo sapiens (human)
- Oxidative phosphorylation - Homo sapiens (human)
- Arginine biosynthesis - Homo sapiens (human)
- Purine metabolism - Homo sapiens (human)
- Caffeine metabolism - Homo sapiens (human)
- Pyrimidine metabolism - Homo sapiens (human)
- Alanine, aspartate and glutamate metabolism - Homo sapiens
- Glycine, serine and threonine metabolism - Homo sapiens (hu
- Cysteine and methionine metabolism - Homo sapiens (human)
- Valine, leucine and isoleucine degradation - Homo sapiens (hu
- Valine, leucine and isoleucine biosynthesis - Homo sapiens (h
- Lysine degradation - Homo sapiens (human)
- Arginine and proline metabolism - Homo sapiens (human)
- Histidine metabolism - Homo sapiens (human)
- Tyrosine metabolism - Homo sapiens (human)
- Phenylalanine metabolism - Homo sapiens (human)

Figure 24: DSA Database Manager



Please do NOT add tampered kgml files.

When adding diseases from Kyoto Encyclopedia of Genes and Genomes(KEGG), please note that NOT all of the pathways listed are disease pathways.

Currently Stored Disease Pathways

- Bladder cancer
- Colorectal cancer

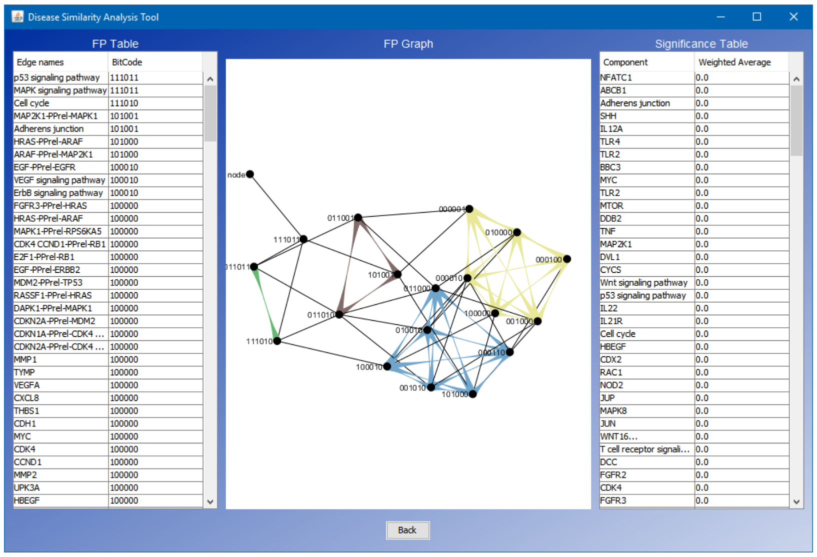
- hsa05210.xml
- hsa05212.xml
- hsa05216.xml
- hsa05219.xml
- hsa05226.xml
- hsa05321.xml

File name: hsa05210.xml

Files of type: All Files

Figure 25: DSA Adding KGML from local file system

In the Disease Search and Subgraph Search functionalities, there is an option to view the FP-Graph and other details that is used by the FP-GraphMiner algorithm to perform the aforementioned functionalities. Clicking this option will display a window as shown in Figure 26.

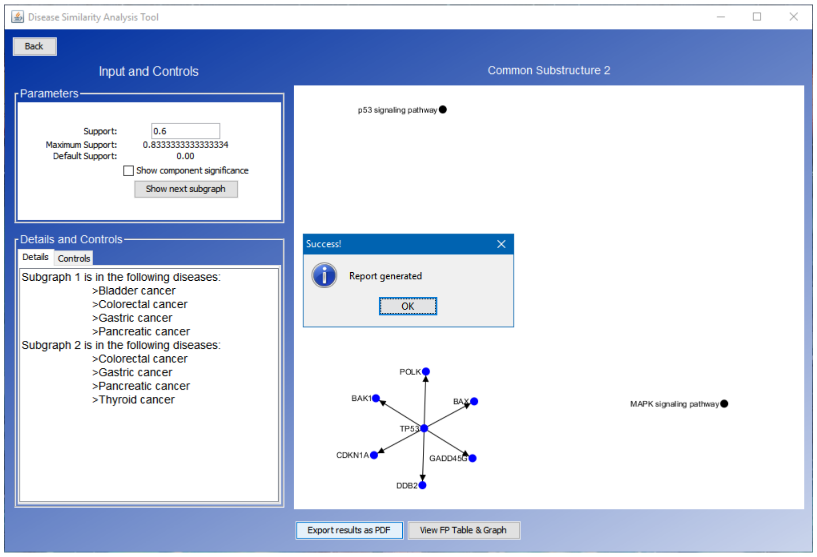


Edge names	BitCode
p53 signaling pathway	111011
MAPK signaling pathway	111011
Cell cycle	111010
MAP2K1-PPrel-MAPK1	101001
Adherens junction	101001
HRAS-PPrel-ARAF	101000
ARAF-PPrel-MAP2K1	101000
EGF-PPrel-EGFR	100010
VEGF signaling pathway	100010
ErbB signaling pathway	100010
FGFR3-PPrel-HRAS	100000
HRAS-PPrel-ARAF	100000
MAPK1-PPrel-RPS6KA5	100000
CDK4 CCND1-PPrel-RB1	100000
E2F1-PPrel-RB1	100000
EGF-PPrel-ERBB2	100000
MDM2-PPrel-TP53	100000
RASSF1-PPrel-HRAS	100000
DAPK1-PPrel-MAPK1	100000
CDKN2A-PPrel-MDM2	100000
CDKN1A-PPrel-CDK4 ...	100000
CDKN2A-PPrel-CDK4 ...	100000
MMP1	100000
TYMP	100000
VEGFA	100000
CXCL8	100000
THBS1	100000
CDH1	100000
MYC	100000
CDK4	100000
CCND1	100000
MMP2	100000
UPK3A	100000
HBEGF	100000

Component	Weighted Average
NFATC1	0.0
ABCB1	0.0
Adherens junction	0.0
SHH	0.0
IL12A	0.0
TLR4	0.0
TLR2	0.0
BBC3	0.0
MYC	0.0
TLR2	0.0
MTOR	0.0
DDB2	0.0
TNF	0.0
MAP2K1	0.0
DVL1	0.0
CYCS	0.0
Wnt signaling pathway	0.0
p53 signaling pathway	0.0
IL22	0.0
IL21R	0.0
Cell cycle	0.0
HBEGF	0.0
CDX2	0.0
RAC1	0.0
NOD2	0.0
JUP	0.0
MAPK8	0.0
JUN	0.0
WNT16...	0.0
T cell receptor signali...	0.0
DCC	0.0
FGFR2	0.0
CDK4	0.0
FGFR3	0.0

Figure 26: DSA FP-GraphMiner Window

In the analysis functionalities, the user is allowed to export the results as a pdf file. A notification, as shown in Figure 27 will tell the user that the report which can be found in the directory of this tool's jar file was successfully generated.



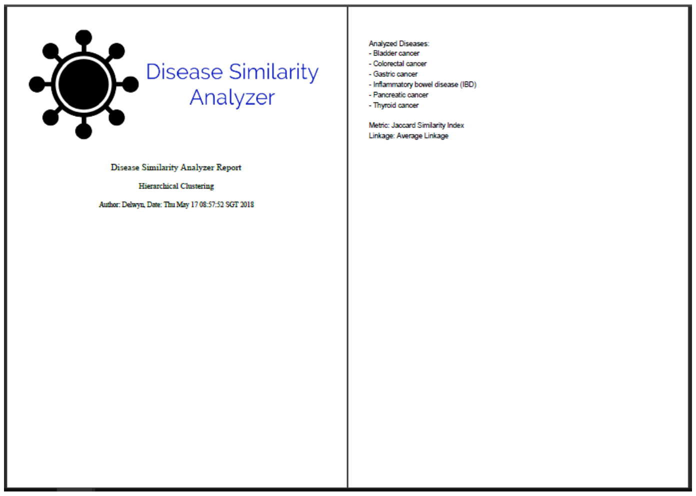
Support: 0.6
Maximum Support: 0.8333333333333334
Default Support: 0.00
☐ Show component significance

Subgraph 1 is in the following diseases:
>Bladder cancer
>Colorectal cancer
>Gastric cancer
>Pancreatic cancer
Subgraph 2 is in the following diseases:
>Colorectal cancer
>Gastric cancer
>Pancreatic cancer
>Thyroid cancer

Report generated

Figure 27: DSA Export notification

The following figures show a sample report that was generated using of the analysis functionalities of this tool. Figure 28 shows the title page and the detail page while the following pages, shown by Figure 29 shows the output of the hierarchical clustering of the tool.



Disease Similarity Analyzer

Disease Similarity Analyzer Report

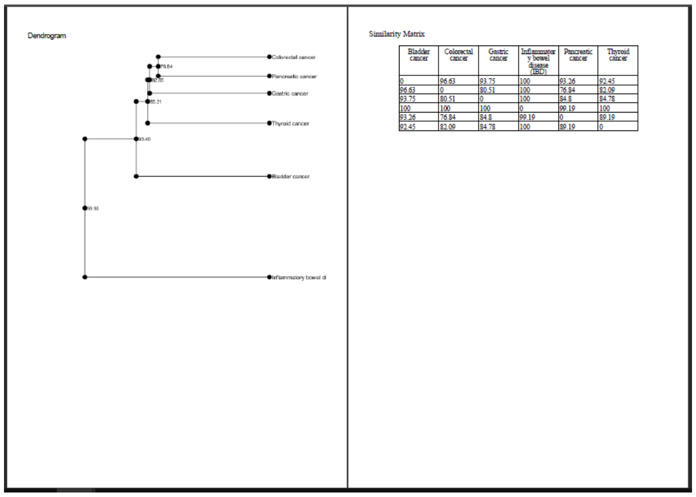
Hierarchical Clustering

Author: Delwyn, Date: Thu May 17 08:57:52 SGT 2018

Analyzed Diseases:
- Bladder cancer
- Colorectal cancer
- Gastric cancer
- Inflammatory bowel disease (IBD)
- Pancreatic cancer
- Thyroid cancer

Metric: Jaccard Similarity Index
Linkage: Average Linkage

Figure 28: DSA PDF Title and Details (Hierarchical Clustering)



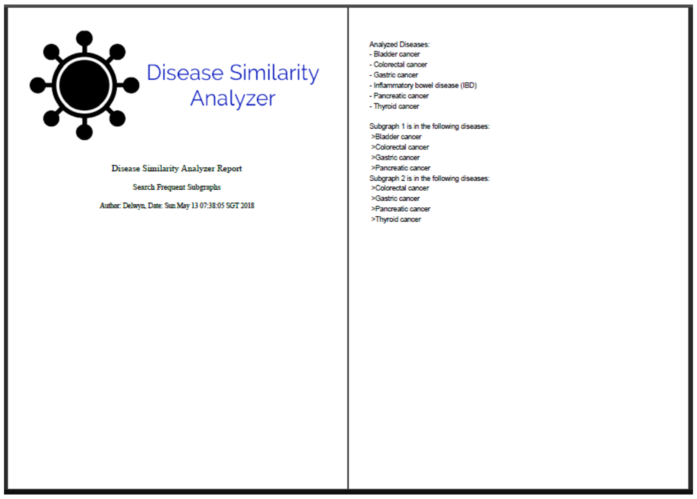
Dendrogram

Similarity Matrix

Bladder cancer	Colorectal cancer	Gastric cancer	Inflammatory bowel disease (IBD)	Pancreatic cancer	Thyroid cancer
0	96.63	93.75	100	93.26	92.45
96.63	0	80.51	100	76.84	82.09
93.75	80.51	0	100	84.8	84.78
100	100	100	0	99.19	100
93.26	76.84	84.8	99.19	0	89.19
92.45	82.09	84.78	100	89.19	0

Figure 29: DSA PDF Analysis Output (Hierarchical Clustering)

Figures 30 and 31 shows a sample report generated by the frequent substructure search functionality of the tool.


Disease Similarity Analyzer

Disease Similarity Analyzer Report

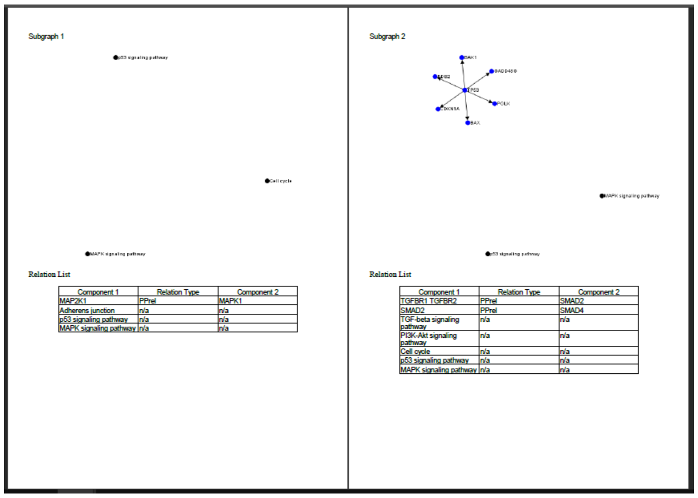
Search Frequent Subgraphs

Author: Delwyn, Date: Sun May 13 07:38:05 SGT 2018

Analyzed Diseases:
- Bladder cancer
- Colorectal cancer
- Gastric cancer
- Inflammatory bowel disease (IBD)
- Pancreatic cancer
- Thyroid cancer

Subgraph 1 is in the following diseases:
>Bladder cancer
>Colorectal cancer
>Gastric cancer
>Pancreatic cancer
Subgraph 2 is in the following diseases:
>Colorectal cancer
>Gastric cancer
>Pancreatic cancer
>Thyroid cancer

Figure 30: DSA PDF Analysis Output (Frequent Substructure Search)



Subgraph 1

Relation List

Component 1	Relation Type	Component 2
MAP2K1	PPrel	MAPK1
Adherens junction	n/a	n/a
p53 signaling pathway	n/a	n/a
MAPK signaling pathway	n/a	n/a

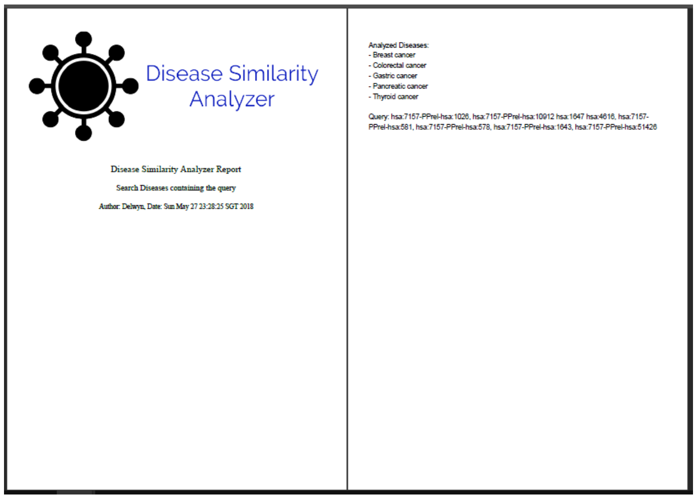
Subgraph 2

Relation List

Component 1	Relation Type	Component 2
TGFBR1 TGFBR2	PPrel	SMAD2
SMAD2	PPrel	SMAD4
TGF-beta signaling pathway	n/a	n/a
PI3K-Akt signaling pathway	n/a	n/a
Cell cycle	n/a	n/a
p53 signaling pathway	n/a	n/a
MAPK signaling pathway	n/a	n/a

Figure 31: DSA PDF Analysis Output (Frequent Substructure Search)

Lastly, figures 32 and 33 shows a sample report generated by the disease search functionality of the tool.



Disease Similarity Analyzer

Disease Similarity Analyzer Report

Search Diseases containing the query

Author: Delwyn, Date: Sun May 27 23:28:25 SGT 2018

Analyzed Diseases:
- Breast cancer
- Colorectal cancer
- Gastric cancer
- Pancreatic cancer
- Thyroid cancer

Query: hsa:7157-PPrel-hsa:1026, hsa:7157-PPrel-hsa:10912 hsa:1647 hsa:4616, hsa:7157-PPrel-hsa:581, hsa:7157-PPrel-hsa:578, hsa:7157-PPrel-hsa:1643, hsa:7157-PPrel-hsa:51426

Figure 32: DSA PDF Analysis Output (Disease Search)

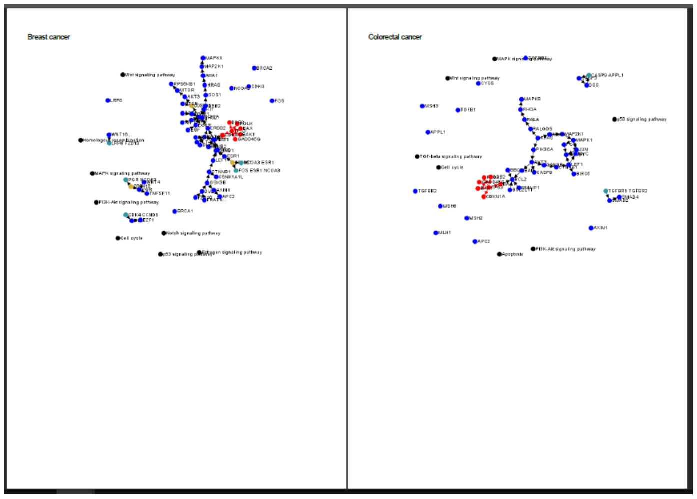


Figure 33: DSA PDF Analysis Output (Disease Search)

VI. Discussions

Disease Similarity Analyzer (DSA) is a Java application that aims to derive similarity as well as mine and vizualize the frequent substructures in human diseases based on their disease pathways from Kyoto Encyclopedia of Genes and Genomes (KEGG).

This application provides its users three main disease similarity analysis functions namely; Hierarchical Clustering, Disease Search and Frequent Substructure search. In each of these functionalities, the disease pathways are represented as graphs which is then used by the two algorithms in this tool: Agglomerative Hierarchical Clustering and FP-GraphMiner.

Using DSA, its users can compute the similarity among the selected disease pathways using agglomerative hierarchical clustering based on the selected metric and linkage type. The resulting similarity is then displayed by this tool using a dendrogram and a similarity matrix. Through the use of FP-GraphMiner algorithm on the graph representation of the disease pathways, the user can either search for diseases containing the specified set of relations or get the frequent substructures present in the selected diseases based on the user defined support value. This tool uses GraphStream library in visualizing the output set of diseases or set of frequent substructures. Using this library enables convenient visualization, especially when dealing with complex graphs such as disease pathways as it allows the users to highlight the query, zoom, pan and move around the nodes of the graph.

The results of the different similarity and analysis functions provided by this tool are supported by the current knowledge regarding human disease structure and their verified shared components. For example, performing hierarchical clustering on the disease set {Asthma, Bladder Cancer, Breast Cancer, Colorectal Cancer, Gastric Cancer, Hepatitis B, Hepatitis C, Inflammatory bowel disease (IBD), Pancreatic Cancer, Thyroid Cancer} shows that the various types of cancers are clustered close to each other. This is supported by the current knowledge that these cancers have the same components such as TP53[22] gene and more importantly, p53 signaling pathway which is currently being targeted to help in cancer medication[23]. These shared components can be then mined and viewed using the remaining functionalities of this tool.

Disease Similarity Analyzer has other functionalities such as importing kgml files from the computer's file system or the KEGG Pathway database using REST API, exporting the analysis outputs in PDF format and viewing a simple user guide.

Unlike other disease similarity analysis methods which mainly use statistics in calculating disease similarity, Disease Similarity Analyzer uses graph representation of disease of disease pathways. This tool also uses FP-GraphMiner algorithm in acquiring the frequent substructures in the selected disease, a method not yet used by the currently existing human disease pathways similarity tools. This algorithm may provide new frequent substructures that may be considered in human disease studies as well as support to the current knowledge about the said diseases.

However using this tool has several drawbacks. One of these is that the similarity that is computed by this tool is strictly based only on the graph representation of the disease pathways. Another drawback of this tool would be its dependence to the Kyoto Encyclopedia of Genes and Genomes (KEGG), specifically to its kgml files, therefore limiting the flexibility of disease pathway data that can be used by this tool.

VII. Conclusions

Disease Similarity Analyzer is a stand-alone application that was developed to allow its users to establish similarity and acquire frequent substructures among selected human diseases using their disease pathways from Kyoto Encyclopedia of Genes and Genomes(KEGG). Through the use of Jaccard Similarity Index and Hamming Distance as similarity metrics and Agglomerative hierarchical clustering, the users can compute and visualize the similarity among the selected diseases in the form of a dendrogram. This tool also allows its users to search for frequent substructures that exists on a user specified percentage of the selected disease pathways using FP-GraphMiner algorithm. Furthermore, using the same algorithm, this tool allows its users to search for diseases containing a specified set of pathway component relation.

Disease Similarity Analyzer aims to assist researchers to establish disease similarity based on their disease pathways that may enable them to support or give new insights about the current knowledge about disease similarity. This tool is also expected to help researchers in their disease studies by providing a way to acquire and visualize frequent substructures in the disease pathways of their specified disease set.

VIII. Recommendations

Disease Similarity Analyzer is a working tool that can be used to cluster and analyze common entities or substructures in human diseases but it's functionalities can still be improved.

One of the improvements that can be implemented is giving the users more options in the linkage type to be used aside from the basic linkages that the tool currently has. Another improvement that can be applied in the clustering functionality would be a better visualization of the generated dendrogram.

The visualizations of frequent substructure search and relation search can also be improved by displaying more data regarding the components and their relations without making the display look convoluted. This would help the researchers in studying the analysis outputs that the tool provides.

Finally, the FP-GraphMiner display may also be improved by showing the paths or steps that the algorithm made to deliver its analysis outputs. By showing the steps together with the currently display FP-Graph and FP-Table, the users would be able to know the process behind the analysis and hopefully, find some improvements to optimize the procedure that is currently being done by the tool.

IX. Bibliography

[1] J. Yang, S.-J. Wu, S.-Y. Yang, J.-W. Peng, S.-N. Wang, F.-Y. Wang, Y.-X. Song, T. Qi, Y.-X. Li, and Y.-Y. Li, "DNetDB: The human disease network database based on dysfunctional regulation mechanism," *BMC Systems Biology*, 2016.

[2] J.-A. R. Sarmiento, A. Lao, and G. A. Solano, "Pathway-based Human Disease Clustering Tool using Self-Organizing Maps," *Information, Intelligence, Systems and Applications (IISA)*, 2017.

[3] J. Thomas, D. Seo, and L. Sael, "Review on Graph Clustering and Subgraph Similarity Based Analysis of Neurological Disorders," *International Journal of Molecular Sciences*, 2016.

[4] T. Theodosiou, G. Efstathiou, N. Papanikolaou, N. C. Kyrpides, P. G. Bagos, I. Iliopoulos, and G. A. Pavlopoulos, "NAP: The Network Analysis Profiler, a web tool for easier topological analysis and comparison of medium-scale biological networks," *BMC Research Notes*, 2017.

[5] X. Z. Zhou, J. Menche, A.-L. Barabasi, and A. Sharma, "Human symptoms-disease network," *Nature communications*, 2014.

[6] K. Sun, J. Goncalves, C. Larminie, and N. Przulj, "Predicting disease associations via biological network analysis," *BMS Bioinformatics*, 2014.

[7] "KEGG overview." `http://www.kegg.jp/kegg/kegg1a.html`. Accessed: 2017-12-13.

[8] R. Vijayalakshmi, N. Rethnasamy, J. F. Roddick, M. Thilaga, and P. Nirmala, "FP-GraphMiner-a fast frequent pattern mining algorithm for network graphs," *J. Graph Algorithms Appl.*, vol. 15, pp. 753–776, 2011.

[9] J. Ren, H. Wang, Y. M. anf Yanling Li, and J. Dong, “Efficient software fault localization by hierarchical instrumentation and maximal frequent subgrah mining,” *International Journal of Innovative Computing, Innovation and Control*, 2015.

[10] L. C. L. Osorio, M. C. O. Carillo, G. Solano, and H. N. Adorna, “Generating phenograms using frequent structure mining over metabolic pathways,” *Information, Intelligence, Systems and Applications (IISA)*, 2015.

[11] C. Wang, W.-H. Kao, and C. K. Hsiao, “Using hamming distance as information for snp-sets clustering and testing in disease association studies,” *PLoS ONE*, 2015.

[12] L. Cheng, J. Li, J. Peng, and Y. Wang, “Semfunsim: A new method for measuring disease similarity by integrating semantic and gene functional association,” *PLoS ONE*, 2014.

[13] D.-H. Le, B.-S. Pham, and A.-M. Dao, “Assessing human disease phenotype similarity based on ontology,” *International Conference on Computing and Communication Technologies, Research, Innovation and Vision for the Future (IEEE RIVF)*, 2016.

[14] Y. Li and P. Agarwal, “A pathway-based view of human diseases and disease relationships,” *PLoS ONE*, 2009.

[15] D. P. Dabhi and M. R. Patel, “Extensive survey on hierarchical clustering methods in data mining,” *International Research Journal of Engineering and Technology*, 2016.

[16] E. G. III, G. Solano, J. Clemente, and H. Adorna, “Building phylogenetic trees from frequent subgraph mining techniques on reaction hypergraphs,” *Proceedings*

of the 5th IEEE International Conference on Information, Intelligence, Systems and Applications, 2014.

[17] H. Dinari and H. Naderi, "A survey of frequent subgraphs and subtree mining methods," *International Journal of Computer Science and Business Informatics*, 2014.

[18] M. Thilaga, R. Vijayalakshmi, and R. N. anf D. Nandagopal, "A novel pattern mining approach for identifying cognitive activity in eeg based functional brain networks," *Journal of Integrative Neuroscience*, 2016.

[19] "KGML (KEGG Markup Language)." `http://www.kegg.jp/kegg/xml/l`. Accessed: 2017-12-13.

[20] "KEGG disease database." `http://www.genome.jp/kegg/disease/`. Accessed: 2017-12-13.

[21] S.-S. Choi, S.-H. Cha, and C. C. Tappert, "A survey of binary similarity and distance measures," *Systemics, Cybernetics and Informatics*, 2010.

[22] "TP53 gene genetic home reference." `https://ghr.nlm.nih.gov/gene/TP53#conditions`. Accessed: 2018-05-20.

[23] A. H. Stegh, "Targeting the p53 signaling pathway in cancer therapy - the promises, challenges, and perils," *National Center for Biotechnology Information-PubMed Central*, 2012.

X. Appendix

A. Source Code

FPNode.java

```
package fpgraphminer;

import java.util.ArrayList;

public class FPNode {

        private ArrayList<String[]> tuples;
        private ArrayList<FPNode> connectedNodes;
        private String bitcode;
        private int weight;
        private double support;
        private boolean isHeader;

        public FPNode(){
                this.tuples = new ArrayList<String[]>();
                this.connectedNodes = new ArrayList<FPNode>();
                this.isHeader = false;
        }

        public void addTuple(String[] tuple){
                this.tuples.add(tuple);
                this.weight = computeWeight(tuple[1]);
                this.bitcode = tuple[1];
                this.support = (double)this.weight/(double)this.bitcode.length();
        }

        public void setBitcode(String bitcode){
                this.bitcode = bitcode;
        }

        public void connectNode(FPNode node){
                this.connectedNodes.add(node);
        }

        public void setAsHeader(){
                this.isHeader = true;
        }

        public int getWeight(){
                return this.weight;
        }

        public ArrayList<FPNode> getConnectedNodes(){
                return this.connectedNodes;
        }

        public double getSupport(){
                return this.support;
        }

        public String getBitcode(){
                return this.bitcode;
        }

        public ArrayList<String[]> getTuples(){
                return this.tuples;
        }

        public boolean isHeader(){
                return this.isHeader;
        }

        public boolean isConnectedtoHeader(){
                for(FPNode neighbor : this.connectedNodes){
                        if(neighbor.isHeader()){
                                return true;
                        }
                }
                return false;
        }

        private int computeWeight(String bitcode){
                int weight = 0;
                for(int counter=0; counter<bitcode.length(); counter++){
                        if(bitcode.charAt(counter) == '1'){
                                weight++;
                        }
                }
                return weight;
        }

}
```

FPCluster.java

```
package fpgraphminer;

import java.util.ArrayList;

public class FPCluster {

        private ArrayList<FPNode> nodeList;
        private int weight;
        private double support;

        public FPCluster(){
                this.nodeList = new ArrayList<FPNode>();
                this.weight = 0;
                this.support = 0;
        }

        public void addNode(FPNode node){
                this.nodeList.add(node);
                if(this.nodeList.size()==1){
                        this.support = node.getSupport();
                        this.weight = node.getWeight();
                }
        }

        public ArrayList<FPNode> getNodes(){
                return this.nodeList;
        }

        public double getSupport(){
                return this.support;
        }

        public int getWeight(){
                return this.weight;
        }
}
```

FPGraph.java

```
package fpgraphminer;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Hashtable;

import model.Disease;
import model.Component;

public class FPGraph {

        private ArrayList<Disease> graphDB = new ArrayList<Disease>();
        private FPNode headerNode;
        private ArrayList<String[]> fptable;
        private ArrayList<String> commonEdgeName, commonCompName;
        private Hashtable<String,String> encoding;

        public FPGraph(ArrayList<Disease> graphDB){
                this.graphDB = graphDB;
                this.commonCompName = new ArrayList<String>();
                this.commonEdgeName = new ArrayList<String>();
                this.encoding = new Hashtable<String,String>();
        }

        public ArrayList<String> getDistinctEdges(ArrayList<Disease> graphDB){
                ArrayList<String> distinctEdges = new ArrayList<String>();
                for(int counter=0; counter<graphDB.size(); counter++){
                        for(int relCounter=0; relCounter<graphDB.get(counter).getRelations().size
                             (); relCounter++){
                                if(!distinctEdges.contains(graphDB.get(counter).getRelations().get
                                     (relCounter).getName())){
                                        distinctEdges.add(graphDB.get(counter).getRelations().get(
                                             relCounter).getName());
                                        this.commonEdgeName.add(graphDB.get(counter).getRelations
                                             ().get(relCounter).getCommonName());
                                        this.encoding.put(graphDB.get(counter).getRelations().get(
                                             relCounter).getName(), graphDB.get(counter).
                                             getRelations().get(relCounter).getCommonName());
                                }
                        }
                }
                return distinctEdges;
```

```
}

public ArrayList<String> getDistinctEdgesCommonName(){
        return this.commonEdgeName;
}

public ArrayList<String> getDistinctDisconnectedNodes(ArrayList<Disease> graphDB){
        ArrayList<String> distinctDC = new ArrayList<String>();
        for(Disease disease: this.graphDB){
                for(Component comp : disease.disconnectedComponents()){
                        if(!distinctDC.contains(comp.getName())){
                                distinctDC.add(comp.getName());
                                this.commonCompName.add(comp.getCommonName());
                                this.encoding.put(comp.getName(),comp.getCommonName());
                        }
                }
        }
        return distinctDC;
}

public ArrayList<String> getDCCommonNames(){
        return this.commonCompName;
}

public Hashtable<String,String> getEncoding(){
        return this.encoding;
}

public ArrayList<FPCluster> createFPGraph(ArrayList<String> distinctEdges, ArrayList<
     String> distinctDC){
        ArrayList<String[]> fptable = createFPTable(this.graphDB, distinctEdges,
             distinctDC);
        this.fptable = fptable;
        return makeClusters(makeNodes(fptable));
}

public FPNode getHeaderNode(){
        return this.headerNode;
}

public ArrayList<String[]> getFPTable(){
        return this.fptable;
}

private ArrayList<String[]> createFPTable(ArrayList<Disease> graphDB, ArrayList<String>
     distinctEdges, ArrayList<String> distinctDC){
        ArrayList<String[]> fptable = new ArrayList<String[]>();
        for(int DEctr=0; DEctr<distinctEdges.size(); DEctr++){
                String[] row = {distinctEdges.get(DEctr),""};
                fptable.add(row);
        }
        for(String disconnected:distinctDC){
                String[] row = {disconnected,""};
                fptable.add(row);
        }
        for(int diseaseCtr=0; diseaseCtr<graphDB.size(); diseaseCtr++){ //FOR EACH DISEASE
                ArrayList<String> relationList = graphDB.get(diseaseCtr).getRelationNames
                     (); // GET THE RELATIONS
                ArrayList<String> componentDCnames = graphDB.get(diseaseCtr).
                     disconnectedComponentNames(); // GET DISCONNECTED COMPONENTS
                for(int relationCtr=0; relationCtr<distinctEdges.size(); relationCtr++){
                     // FOR EACH RELATION
                        if(relationList.contains(distinctEdges.get(relationCtr))){ //IF
                             THE RELATION IS IN THE DISTINCT EDGES
                                fptable.get(relationCtr)[1] = fptable.get(relationCtr)
                                     [1]+"1";
                        }else{
                                fptable.get(relationCtr)[1] = fptable.get(relationCtr)
                                     [1]+"0";
                        }
                }
                for(int dcCtr=0; dcCtr<distinctDC.size();dcCtr++){//FOR EACH DISCONNECTED
                     COMPONENT
                        if(componentDCnames.contains(distinctDC.get(dcCtr))){
                                fptable.get(dcCtr+distinctEdges.size())[1] = fptable.get(
                                     dcCtr+distinctEdges.size())[1]+"1";
                        }else{
                                fptable.get(dcCtr+distinctEdges.size())[1] = fptable.get(
                                     dcCtr+distinctEdges.size())[1]+"0";
                        }
                }
        }
        Collections.sort(fptable, new Comparator<String[]>() {
            @Override
            public int compare(String[] bitcode1, String[] bitcode2) {
                return bitcode2[1].compareTo(bitcode1[1]);
            }
        });
        return fptable;
}

public ArrayList<String[]> makeDiseaseBitcodeTable(ArrayList<Disease> graphDB, ArrayList<
    String> distinctEdges, ArrayList<String> distinctDC){
        ArrayList<String[]> table = new ArrayList<String[]>();
        for(Disease disease : graphDB){
                String[] row = {disease.getTitle(),""};
                table.add(row);
        }
        for(int relationCtr=0; relationCtr<distinctEdges.size(); relationCtr++){ // FOR
            EACH RELATION
                for(int diseaseCtr=0; diseaseCtr<graphDB.size(); diseaseCtr++){ //FOR EACH
                        DISEASE
                        ArrayList<String> relationList = graphDB.get(diseaseCtr).
                             getRelationNames(); // GET THE RELATIONS
                                if(relationList.contains(distinctEdges.get(relationCtr))){
                                     //IF THE RELATION IS IN THE DISTINCT EDGES
                                        table.get(diseaseCtr)[1] = table.get(diseaseCtr)
                                             [1]+"1";
                                }else{
                                        table.get(diseaseCtr)[1] = table.get(diseaseCtr)
                                             [1]+"0";
                                }
                        }
        }
        for(int dcCtr=0; dcCtr<distinctDC.size();dcCtr++){//FOR EACH DISCONNECTED
            COMPONENT
                for(int diseaseCtr=0; diseaseCtr<graphDB.size(); diseaseCtr++){ //FOR EACH
                        DISEASE
                        ArrayList<String> componentDCnames = graphDB.get(diseaseCtr).
                             disconnectedComponentNames(); // GET DISCONNECTED COMPONENTS
                        if(componentDCnames.contains(distinctDC.get(dcCtr))){
                                table.get(diseaseCtr)[1] = table.get(diseaseCtr)[1]+"1";
                        }else{
                                table.get(diseaseCtr)[1] = table.get(diseaseCtr)[1]+"0";
                        }
                }
        }
        return table;
}

private ArrayList<FPCluster> makeClusters(ArrayList<FPNode> nodes){
        ArrayList<FPCluster> clustered = new ArrayList<FPCluster>();
        ArrayList<Integer> uniqueWeights = getUniqueWeights(nodes);
        for(int weightCtr=0; weightCtr<uniqueWeights.size(); weightCtr++){
                FPCluster cluster = new FPCluster();
                for(int nodeCtr=0; nodeCtr<nodes.size(); nodeCtr++){
                        if(nodes.get(nodeCtr).getWeight() == uniqueWeights.get(weightCtr))
                             {
                                cluster.addNode(nodes.get(nodeCtr));
                        }
                }
                clustered.add(cluster);
        }
        Collections.sort(clustered, new Comparator<FPCluster>() {
            @Override
            public int compare(FPCluster cluster1, FPCluster cluster2) {
                return Double.compare(cluster1.getSupport(),cluster2.getSupport());
            }
        });
        connectNodes(clustered);
        FPNode headerNode = new FPNode();
        headerNode.setAsHeader();
        for(int nodeCtr=0; nodeCtr<clustered.get(clustered.size()-1).getNodes().size();
            nodeCtr++){
                headerNode.connectNode(clustered.get(clustered.size()-1).getNodes().get(
                     nodeCtr));
                clustered.get(clustered.size()-1).getNodes().get(nodeCtr).connectNode(
                     headerNode);
        }
        this.headerNode = headerNode;
        return clustered;
}

private ArrayList<Integer> getUniqueWeights(ArrayList<FPNode> nodes){
        ArrayList<Integer> uniqueWeights = new ArrayList<Integer>();
        for(int nodeIter=0; nodeIter<nodes.size(); nodeIter++){
                if(!uniqueWeights.contains(nodes.get(nodeIter).getWeight())){
                        uniqueWeights.add(nodes.get(nodeIter).getWeight());
                }
        }
        return uniqueWeights;
}

private ArrayList<FPNode> makeNodes(ArrayList<String[]> fptable){
        ArrayList<FPNode> nodes = new ArrayList<FPNode>();
        ArrayList<String> bitcodes = getUniqueBitcodes(fptable);
        for(String bitcode : bitcodes){
                FPNode node = new FPNode();
                for(String[] tuple : fptable){
                        if(tuple[1].equals(bitcode)){
                                node.addTuple(tuple);
                        }
                }
                nodes.add(node);
    }
    return nodes;
}

private ArrayList<String> getUniqueBitcodes(ArrayList<String[]> fptable){
    ArrayList<String> bitcodes = new ArrayList<String>();
    for(int rowCtr=0; rowCtr<fptable.size(); rowCtr++){
        if(!bitcodes.contains(fptable.get(rowCtr)[1])){
            bitcodes.add(fptable.get(rowCtr)[1]);
        }
    }
    return bitcodes;
}

private void connectNodes(ArrayList<FPCluster> clusters){
    for(int clustCtr=0; clustCtr<clusters.size(); clustCtr++){
        for(FPNode node1 : clusters.get(clustCtr).getNodes()){
            boolean hasSuperset = false;
            for(int nextClustCtr=clustCtr+1; nextClustCtr<clusters.size();
                nextClustCtr++){
                if(!hasSuperset){
                    for(FPNode node2 : clusters.get(nextClustCtr).
                        getNodes()){
                        if(AND(node1.getBitcode(),node2.getBitcode
                            ()).equals(node1.getBitcode())){
                            if(!node1.getConnectedNodes().
                                contains(node2)){
                                node1.connectNode(node2);
                            }
                            if(!node2.getConnectedNodes().
                                contains(node1)){
                                node2.connectNode(node1);
                            }
                            hasSuperset = true;
                        }
                    }
                }
            }
        }
    }
}

private String AND(String bitcode1, String bitcode2){
    String result = "";
    for(int charCtr=0; charCtr<bitcode1.length(); charCtr++){
        if(bitcode1.charAt(charCtr) == '1' && bitcode2.charAt(charCtr) == '1'){
            result = result+"1";
        }else{
            result = result+"0";
        }
    }
    return result;
}

}
```

FPGraphMiner.java

```
package fpgraphminer;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Hashtable;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Set;


public class FPGraphMiner {

    public ArrayList<ArrayList<FPNode>> findFrequentSubgraphs(ArrayList<FPCluster> fpgraph,
        double support){
        ArrayList<ArrayList<FPNode>> frequentSubgraphs = new ArrayList<ArrayList<FPNode
            >>();
        FPCluster startingCluster = null;
        for(int clustCtr=0; clustCtr<fpgraph.size(); clustCtr++){
            if(fpgraph.get(clustCtr).getSupport()>=support){
                startingCluster = fpgraph.get(clustCtr);
                break;
            }
        }
        for(int nodeCtr=0; nodeCtr<startingCluster.getNodes().size(); nodeCtr++){
            ArrayList<FPNode> visited = new ArrayList<FPNode>();
            ArrayList<FPNode> path = new ArrayList<FPNode>();
            dfsMiner(startingCluster.getNodes().get(nodeCtr), visited, path, support,
                frequentSubgraphs);
        }
        return frequentSubgraphs;
    }
```

```
@SuppressWarnings("unchecked")
private void dfsMiner(FPNode startingNode, ArrayList<FPNode> visited, ArrayList<FPNode>
     path, double support, ArrayList<ArrayList<FPNode>> frequentSubgraphs){
        visited.add(startingNode);
        path.add(startingNode);
        if(startingNode.isConnectedtoHeader()){
                frequentSubgraphs.add((ArrayList<FPNode>)path.clone());
        }else{
                for(int neighborCtr=0; neighborCtr<startingNode.getConnectedNodes().size()
                     ; neighborCtr++){
                        if(!visited.contains(startingNode.getConnectedNodes().get(
                             neighborCtr))){
                                if(startingNode.getConnectedNodes().get(neighborCtr).
                                     getSupport()>=support){
                                        dfsMiner(startingNode.getConnectedNodes().get(
                                             neighborCtr), visited, path, startingNode.
                                             getConnectedNodes().get(neighborCtr).
                                             getSupport(), frequentSubgraphs);
                                }
                        }
                }
        }
        path.remove(path.size()-1);
        visited.remove(startingNode);
}

public String diseaseFinder(ArrayList<String> relationNames, FPNode headerNode){
        ArrayList<FPNode> visited = new ArrayList<FPNode>();
        ArrayList<String> bitcodes = new ArrayList<String>();
        ArrayList<Boolean> marker = new ArrayList<Boolean>();
        Queue<FPNode> bfsQueue = new LinkedList<>();
        bfsQueue.add(headerNode);
        visited.add(headerNode);
        for(int markCtr=0; markCtr<relationNames.size(); markCtr++){
                marker.add(false);
        }
        while(!bfsQueue.isEmpty()){
                FPNode node = bfsQueue.poll();
                for(FPNode connectedNode : node.getConnectedNodes()){
                        if(!visited.contains(connectedNode)){
                                bfsQueue.add(connectedNode);
                                visited.add(connectedNode);
                                for(String[] tuple : connectedNode.getTuples()){
                                        if(marker.contains(false)){
                                                if(relationNames.contains(tuple[0]) && !
                                                     bitcodes.contains(tuple[1])){
                                                        bitcodes.add(tuple[1]);
                                                        marker.set(relationNames.indexOf(
                                                             tuple[0]), true);
                                                }
                                        }else{
                                                break;
                                        }
                                }
                        }
                }
        }
        String graphBitCode = andOperation(bitcodes);
        return graphBitCode;
}

public Hashtable<String, Double> getWeightedAverages(ArrayList<FPCluster> fpgraph){
        Hashtable<String, Double> weightedAverages = new Hashtable<String, Double>();
        ArrayList<String> uniqueComponents = getUniqueComponents(fpgraph);
        int maxDegree = uniqueComponents.size()-1;
        for(String component : uniqueComponents){
                weightedAverages.put(component, 0.0);
        }
        for(FPCluster cluster : fpgraph){
                ArrayList<Double> counter = new ArrayList<Double>(Collections.nCopies(
                     uniqueComponents.size(), 0.0));
                for(FPNode node : cluster.getNodes()){
                        for(String[] tuple : node.getTuples()){
                                if(tuple[0].indexOf('-')!=-1 && tuple[0].lastIndexOf('-')
                                     !=-1){
                                        String comp1 = tuple[0].substring(0, tuple[0].
                                             indexOf('-'));
                                        String comp2 = tuple[0].substring(tuple[0].
                                             lastIndexOf('-')+1,tuple[0].length());
                                        counter.set(uniqueComponents.indexOf(comp1),
                                             counter.get(uniqueComponents.indexOf(comp1))
                                             +1);
                                        counter.set(uniqueComponents.indexOf(comp2),
                                             counter.get(uniqueComponents.indexOf(comp2))
                                             +1);
                                }else{
                                        counter.set(uniqueComponents.indexOf(tuple[0]),
                                             counter.get(uniqueComponents.indexOf(tuple[0])
                                             )+1);
                                }
                        }
                }
```

```
                        for(int countCtr=0; countCtr<counter.size(); countCtr++){
                                Double currWeight = (counter.get(countCtr)*cluster.getSupport()
                                      *100)/maxDegree;
                                counter.set(countCtr,currWeight);
                                weightedAverages.put(uniqueComponents.get(countCtr),
                                      weightedAverages.get(uniqueComponents.get(countCtr))+
                                      currWeight);
                        }
                }
                Set<String> keys = weightedAverages.keySet();
                for(String key:keys){
                        weightedAverages.replace(key, Math.floor(weightedAverages.get(key)));
                }
                return weightedAverages;
        }

        private ArrayList<String> getUniqueComponents(ArrayList<FPCluster> fpgraph){
                ArrayList<String> uniqueComp = new ArrayList<String>();
                for(FPCluster cluster : fpgraph){
                        for(FPNode node : cluster.getNodes()){
                                for(String[] tuple : node.getTuples()){
                                        if(tuple[0].indexOf('-')!=-1 && tuple[0].lastIndexOf('-')
                                              !=-1){
                                                String comp1 = tuple[0].substring(0, tuple[0].
                                                      indexOf('-'));
                                                String comp2 = tuple[0].substring(tuple[0].
                                                      lastIndexOf('-')+1,tuple[0].length());
                                                if(!uniqueComp.contains(comp1)){
                                                        uniqueComp.add(comp1);
                                                }
                                                if(!uniqueComp.contains(comp2)){
                                                        uniqueComp.add(comp2);
                                                }
                                        }else{
                                                if(!uniqueComp.contains(tuple[0])){
                                                        uniqueComp.add(tuple[0]);
                                                }
                                        }
                                }
                        }
                }
                return uniqueComp;
        }

        private String andOperation(ArrayList<String> bitcodes){
                String result = "";
                for(int charCtr=0; charCtr<bitcodes.get(0).length(); charCtr++){
                        boolean hasZero = false;
                        for(int bitCodeCtr=0; bitCodeCtr<bitcodes.size(); bitCodeCtr++){
                                if(bitcodes.get(bitCodeCtr).charAt(charCtr) == '0'){
                                        hasZero = true;
                                }
                        }
                        if(hasZero){
                                result = result+"0";
                        }else{
                                result = result+"1";
                        }
                }
                return result;
        }

}
```

FPGfunctions.java

```
package fpgraphminer;

import java.util.ArrayList;
import java.util.Hashtable;

import org.graphstream.graph.EdgeRejectedException;
import org.graphstream.graph.Graph;
import org.graphstream.graph.Node;
import org.graphstream.graph.implementations.MultiGraph;
import model.Disease;

public class FPGfunctions {

        private ArrayList<Disease> diseases;
        private FPGraph fpgraph;
        private ArrayList<String> distinctEdges,distinctDC;
        private ArrayList<FPCluster> fpg;
        private ArrayList<ArrayList<String>> relationTable;
        private int subgraphCount;
        private double maxSupport;
        private FPGraphMiner miner;
        private boolean showSignif;

        public FPGfunctions(ArrayList<Disease> selectedDiseases){
```

```
        this.diseases = selectedDiseases;
        this.fpgraph = new FPGraph(this.diseases);
        this.distinctEdges = this.fpgraph.getDistinctEdges(this.diseases);
        this.distinctDC = this.fpgraph.getDistinctDisconnectedNodes(this.diseases);
        this.fpg = this.fpgraph.createFPGraph(this.distinctEdges,this.distinctDC);
        this.relationTable = new ArrayList<ArrayList<String>>();
        this.maxSupport = fpg.get(fpg.size()-1).getSupport();
        this.miner = new FPGraphMiner();
}

public int getSubgraphCount(){
        return this.subgraphCount;
}

public double getMaxSupport(){
        return this.maxSupport;
}

public ArrayList<ArrayList<String>> getRelationTable(){
        return this.relationTable;
}

public Graph showFrequentSubgraphs(ArrayList<ArrayList<FPNode>> frequentSubgraphs, int
    subGraphID){
        Graph graph = new MultiGraph("Graph");
        ArrayList<String> components = new ArrayList<String>();
        ArrayList<String> tupleNames = new ArrayList<String>();
        Hashtable<String,String> encoding = distinctCompEncoding();
        for(int nodeCtr=0; nodeCtr<frequentSubgraphs.get(subGraphID).size(); nodeCtr++){
                for(int tupleCtr=0; tupleCtr<frequentSubgraphs.get(subGraphID).get(nodeCtr
                    ).getTuples().size(); tupleCtr++){
                        String tupleName = frequentSubgraphs.get(subGraphID).get(nodeCtr).
                            getTuples().get(tupleCtr)[0];
                        if(tupleName.indexOf('-')!=-1 && tupleName.lastIndexOf('-')!=-1){
                                String comp1 = tupleName.substring(0, tupleName.indexOf
                                    ('-'));
                                String comp2 = tupleName.substring(tupleName.lastIndexOf
                                    ('-')+1,tupleName.length());
                                if(!components.contains(comp1)){
                                        components.add(comp1);
                                }if(!components.contains(comp2)){
                                        components.add(comp2);
                                }
                        }else{
                                if(!components.contains(tupleName)){
                                        components.add(tupleName); //A DISCONNECTED NODE
                                }
                        }
                }
        }
        for(int compCtr=0; compCtr<components.size(); compCtr++){
                graph.addNode(components.get(compCtr));
        }
        for(int nodeCtr=0; nodeCtr<frequentSubgraphs.get(subGraphID).size(); nodeCtr++){
                for(int tupleCtr=0; tupleCtr<frequentSubgraphs.get(subGraphID).get(nodeCtr
                    ).getTuples().size(); tupleCtr++){
                        String tupleName = frequentSubgraphs.get(subGraphID).get(nodeCtr).
                            getTuples().get(tupleCtr)[0];
                        if(!tupleNames.contains(tupleName)){
                                tupleNames.add(tupleName);
                                if(tupleName.indexOf('-')!=-1 && tupleName.lastIndexOf
                                    ('-')!=-1){
                                        String comp1 = tupleName.substring(0, tupleName.
                                            indexOf('-'));
                                        String comp2 = tupleName.substring(tupleName.
                                            lastIndexOf('-')+1,tupleName.length());
                                        Node compX = graph.getNode(comp1);
                                        Node compY = graph.getNode(comp2);
                                        try{
                                                graph.addEdge(tupleName, compX, compY,
                                                    true);
                                        }catch(EdgeRejectedException e){
                                                System.out.println("Edge already exists");
                                                continue;
                                        }
                                }
                        }
                }
        }
        graph.addAttribute("ui.antialias");

        if(showSignif){
                setSignificance(graph);
        }

        for (Node node : graph) {
        node.addAttribute("ui.label", encoding.get(node.getId()));
        if(node.getId().startsWith("hsa")){
                node.addAttribute("ui.style", "fill-color: rgb(0,0,255); text-alignment:
                    at-left;");
        }else if(node.getId().startsWith("cpd")){
                node.addAttribute("ui.style", "fill-color: rgb(213,172,49); text-alignment
                        : at-left;");
            }else if(node.getId().startsWith("group")){
                    node.addAttribute("ui.style", "fill-color: rgb(49,152,159); text-alignment
                        : at-left;");
            }else{
                    node.addAttribute("ui.style", "text-alignment: at-left;");
            }
        }

            return graph;
    }

    public ArrayList<Disease> getGraphsContainingQuery(ArrayList<String> query){
            ArrayList<Disease> result = new ArrayList<Disease>();
            FPGraphMiner graphMiner = new FPGraphMiner();
            FPNode headerNode = this.fpgraph.getHeaderNode();
            char[] bitcodeArr = graphMiner.diseaseFinder(query, headerNode).toCharArray();
            for(int counter=0; counter<bitcodeArr.length; counter++){
                    if(bitcodeArr[counter] == '1'){
                            result.add(this.diseases.get(counter));
                    }
            }
            return result;
    }

    public Graph showQueryGraph(Disease disease, ArrayList<String> query){
            Graph graph = new MultiGraph("Graph");
            ArrayList<String> components = new ArrayList<String>();
            ArrayList<String> tupleNames = new ArrayList<String>();
            Hashtable<String,String> nameEncoding = disease.getCommonNames();
            for(int relationCtr=0; relationCtr<disease.getRelationNames().size(); relationCtr
                ++){
                    String tupleName = disease.getRelationNames().get(relationCtr);
                    String comp1 = tupleName.substring(0, tupleName.indexOf('-'));
                    String comp2 = tupleName.substring(tupleName.lastIndexOf('-')+1,tupleName.
                        length());
                    if(!components.contains(comp1)){
                            components.add(comp1);
                    }if(!components.contains(comp2)){
                            components.add(comp2);
                    }
            }
            for(int compCtr=0; compCtr<components.size(); compCtr++){
                    graph.addNode(components.get(compCtr));
            }
            for(int relationCtr=0; relationCtr<disease.getRelationNames().size(); relationCtr
                ++){
                    String tupleName = disease.getRelationNames().get(relationCtr);
                    if(!tupleNames.contains(tupleName)){
                            tupleNames.add(tupleName);
                            String comp1 = tupleName.substring(0, tupleName.indexOf('-'));
                            String comp2 = tupleName.substring(tupleName.lastIndexOf('-')+1,
                                tupleName.length());
                            Node compX = graph.getNode(comp1);
                            Node compY = graph.getNode(comp2);
                            try{
                                    graph.addEdge(tupleName, compX, compY, true);
                            }catch(EdgeRejectedException e){
                                    System.out.println("Edge already exists");
                                    continue;
                            }
                            if(query.contains(tupleName)){
                                    graph.getEdge(tupleName).addAttribute("ui.style", "fill-
                                        color: rgb(255,0,0);");
                                    if(!compX.hasAttribute("ui.style")){
                                            compX.addAttribute("ui.style", "fill-color: rgb
                                                (255,0,0); text-alignment: at-left;");
                                    }if(!compY.hasAttribute("ui.style")){
                                            compY.addAttribute("ui.style", "fill-color: rgb
                                                (255,0,0); text-alignment: at-left;");
                                    }
                            }
                    }
            }
            for(int disconnectedCtr=0; disconnectedCtr<disease.disconnectedComponentNames().
                size(); disconnectedCtr++){
                    graph.addNode(disease.disconnectedComponentNames().get(disconnectedCtr));
                    Node node = graph.getNode(disease.disconnectedComponentNames().get(
                        disconnectedCtr));
                    if(query.contains(node.getId())){
                            if(!node.hasAttribute("ui.style")){
                                    node.addAttribute("ui.style", "fill-color: rgb(255,0,0);
                                        text-alignment: at-left;");
                            }
                    }
            }
            graph.addAttribute("ui.antialias");

            if(showSignif){
                    setSignificance(graph);
            }
        for (Node node : graph) {
            node.addAttribute("ui.label", nameEncoding.get(node.getId()));
            if(!node.hasAttribute("ui.style")){
                    if(node.getId().startsWith("hsa")){
                            node.addAttribute("ui.style", "fill-color: rgb(0,0,255); text-
                                alignment: at-left;");
                    }else if(node.getId().startsWith("cpd")){
                            node.addAttribute("ui.style", "fill-color: rgb(213,172,49); text-
                                alignment: at-left;");
                    }else if(node.getId().startsWith("group")){
                            node.addAttribute("ui.style", "fill-color: rgb(49,152,159); text-
                                alignment: at-left;");
                    }else{
                            node.addAttribute("ui.style", "text-alignment: at-left;");
                    }
            }
        }

        return graph;
    }

    public ArrayList<ArrayList<FPNode>> getFrequentSubgraphs(double support){
            FPGraphMiner graphMiner = new FPGraphMiner();
            Hashtable<String,String> encoding = distinctCompEncoding();
            ArrayList<ArrayList<FPNode>> frequentSubgraphs = graphMiner.findFrequentSubgraphs(
                fpg,support);
            this.subgraphCount = frequentSubgraphs.size();
            for(ArrayList<FPNode> subgraph : frequentSubgraphs){
                    ArrayList<String> edges = new ArrayList<String>();
                    for(FPNode node : subgraph){
                            for(String[] tuple : node.getTuples()){
                                    String tupleName = tuple[0];
                                    if(tupleName.indexOf('-')!=-1 && tupleName.lastIndexOf
                                        ('-')!=-1){
                                            String comp1 = tupleName.substring(0, tupleName.
                                                indexOf('-'));
                                            String comp2 = tupleName.substring(tupleName.
                                                lastIndexOf('-')+1,tupleName.length());
                                            String newTupleName = tupleName.replace(comp1,
                                                encoding.get(comp1));
                                            newTupleName = newTupleName.replace(comp2,
                                                encoding.get(comp2));
                                            if(!edges.contains(newTupleName)){
                                                    edges.add(newTupleName);
                                            }
                                    }else{
                                            if(!relationTable.contains(tupleName)){
                                                    edges.add(encoding.get(tupleName));
                                            }
                                    }
                            }
                    }
                    this.relationTable.add(edges);
            }
            return frequentSubgraphs;
    }

    public ArrayList<Graph> getFrequentSubgraphsGraphStream(double support){
            ArrayList<Graph> frequentSubgraphs = new ArrayList<Graph>();
            ArrayList<ArrayList<FPNode>> minedSubgraphs = getFrequentSubgraphs(support);

            for(int ctr = 0; ctr<minedSubgraphs.size(); ctr++){
                    Graph graph = showFrequentSubgraphs(minedSubgraphs,ctr);
                    graph.addAttribute("ui.antialias");
                    frequentSubgraphs.add(graph);
            }

            return frequentSubgraphs;
    }

    public ArrayList<Graph> getDiseasesGraphStream(ArrayList<String> query, ArrayList<Disease>
          diseases){
            ArrayList<Graph> diseaseGraphs = new ArrayList<Graph>();

            for(Disease disease : diseases){
                    Graph diseaseGraph = showQueryGraph(disease,query);
                    diseaseGraph.addAttribute("ui.antialias");
                    diseaseGraphs.add(diseaseGraph);
            }

            return diseaseGraphs;
    }

    public void setSignif(boolean willSet){
            this.showSignif = willSet;
    }

    private void setSignificance(Graph graph){
            graph.addAttribute("ui.stylesheet", "node { size-mode: dyn-size; }");
            Hashtable<String, Double> significanceTable = miner.getWeightedAverages(this.fpg);
            for(Node node : graph){
                    node.addAttribute("ui.size", 10+significanceTable.get(node.getId()));
            }
        }

        public Hashtable<String,String> distinctCompEncoding(){
            Hashtable<String,String> encoding = new Hashtable<String,String>();
            for(Disease disease:this.diseases){
                encoding.putAll(disease.getCommonNames());
            }
            return encoding;
        }

}
```

Cluster.java

```
package hierarchicalClustering;

import java.util.ArrayList;

public class Cluster{

        private Cluster leftChild;
        private Cluster rightChild;
        private Cluster parent;
        private double distance;
        private String name;
        private ArrayList<Cluster> childList;

        public Cluster(double distance, Cluster left, Cluster right){
            this.leftChild = left;
            this.rightChild = right;
            this.parent = null;
            this.distance = distance;
            this.name = "";
            this.childList = new ArrayList<Cluster>();
            this.childList.add(leftChild);
            this.childList.add(rightChild);
        }

        public Cluster(String name){
            this.distance = 0;
            this.name = name;
            this.childList = new ArrayList<Cluster>();
        }

        public boolean hasRightChild(){
            if(this.rightChild != null){
                return true;
            }
            return false;
        }

        public boolean hasLeftChild(){
            if(this.leftChild != null){
                return true;
            }
            return false;
        }

        public Cluster getParent(){
            return this.parent;
        }

        public void setParent(Cluster parent){
            this.parent = parent;
        }

        public void setLeftChild(Cluster left){
            this.leftChild = left;
            this.childList.add(left);
            left.setParent(this);
        }

        public void setRightChild(Cluster right){
            this.rightChild = right;
            this.childList.add(right);
            right.setParent(this);
        }

        public void setDistance(double distance){
            this.distance = distance;
        }

        public void setName(String name){
            this.name = name;
        }

        public Cluster getleftChild(){
            return this.leftChild;
        }
        public Cluster getrightChild(){
                return this.rightChild;
        }

        public double getDistance(){
                return this.distance;
        }

        public String getName(){
                return this.name;
        }

        public ArrayList<Cluster> getChildList(){
                return this.childList;
        }
}
```

SimilarityMatrix.java

```
package hierarchicalClustering;

import java.util.ArrayList;

public class SimilarityMatrix {

        public double[][] makeSimilarityMatrix(ArrayList<String[]> fptable, String metric){
                int deCount = fptable.size();
                double[][] sim = new double[deCount][deCount];
                for(int row=0; row<deCount; row++){
                        double[] rowLine = new double[deCount];
                        for(int col=0; col<deCount; col++){
                                if(metric.equals("Jaccard")){
                                        rowLine[col] = (1-jaccardSimilarity(fptable.get(row)[1],
                                             fptable.get(col)[1]))*100;
                                }
                                if(metric.equals("Hamming")){
                                        rowLine[col] = hammingDistance(fptable.get(row)[1],fptable
                                             .get(col)[1]);
                                }
                        }
                        sim[row] = rowLine;
                }
                return sim;
        }

        private double jaccardSimilarity(String bitcode1, String bitcode2){
                return AND(bitcode1,bitcode2)/OR(bitcode1,bitcode2);
        }

        private double hammingDistance(String bitcode1, String bitcode2){
                double hd = 0;
                for(int c=0; c<bitcode1.length(); c++){
                        if(bitcode1.charAt(c)!=bitcode2.charAt(c)){
                                hd++;
                        }
                }
                return hd;
        }

        private double AND(String bitcode1, String bitcode2){
                double and = 0;
                for(int counter=0; counter<bitcode1.length(); counter++){
                        if(bitcode1.charAt(counter)=='1' && bitcode2.charAt(counter)=='1'){
                                and++;
                        }
                }
                return and;
        }

        private double OR(String bitcode1, String bitcode2){
                double or = 0;
                for(int counter=0; counter<bitcode1.length(); counter++){
                        if(bitcode1.charAt(counter) == '1' || bitcode2.charAt(counter) == '1'){
                                or++;
                        }
                }
                return or;
        }

}
```

ClusterComparator.java

```
package hierarchicalClustering;

import java.util.Comparator;

public class ClusterComparator implements Comparator<Cluster>{
	@Override
	public int compare(Cluster cluster1, Cluster cluster2) {
		return Double.compare(cluster1.getDistance(),cluster2.getDistance());
	}

}
```

AverageLinkage.java

```
package hierarchicalClustering;

import java.util.*;

public class AverageLinkage {

	public Cluster cluster(double[][] distances, ArrayList<String> names, String linkage){
		PriorityQueue<Cluster> clusters = null;
		clusters = new PriorityQueue<Cluster>(new ClusterComparator());
		ArrayList<Cluster> leaves = new ArrayList<Cluster>();
		for(int nameCtr=0; nameCtr<names.size(); nameCtr++){
			Cluster leaf = new Cluster(names.get(nameCtr));
			leaves.add(leaf);
		}
		if(distances.length == names.size()){
			for(int rowCounter=1; rowCounter<distances.length; rowCounter++){
				for(int colCounter=0; colCounter<rowCounter; colCounter++){
					Cluster cluster = new Cluster(distances[rowCounter][
					    colCounter],leaves.get(rowCounter),leaves.get(
					    colCounter));
					cluster.setName(names.get(rowCounter)+"-"+names.get(
					    colCounter));
					clusters.add(cluster);
				}
			}
		}
		while(clusters.size() != 1){
			clusters = recursiveCluster(clusters, linkage);
		}
		return clusters.peek();
	}

	private PriorityQueue<Cluster> recursiveCluster(PriorityQueue<Cluster> clusters, String
	    linkage){
		Cluster minimum = clusters.poll();
		int counter = 0;
		ArrayList<Cluster> toRemove = new ArrayList<Cluster>();
		ArrayList<Cluster> toAdd = new ArrayList<Cluster>();
		for(Cluster clust : clusters){
			if(clust.getChildList().contains(minimum.getleftChild())){
				Cluster common = getOtherChild(clust, minimum.getleftChild());
				double leftDist = clust.getDistance();
				double rightDist = 0;
				Cluster newCluster = new Cluster("cluster"+counter);
				newCluster.setLeftChild(common);
				newCluster.setRightChild(minimum);
				toRemove.add(clust);
				for(Cluster pair : clusters){
					if(pair.getChildList().contains(minimum.getrightChild())
					    && pair.getChildList().contains(common)){
						rightDist = pair.getDistance();
						toRemove.add(pair);
					}
				}
				if(linkage.equals("min")){
					newCluster.setDistance(getMin(leftDist,rightDist));
				}else if(linkage.equals("max")){
					newCluster.setDistance(getMax(leftDist,rightDist));
				}else{
					newCluster.setDistance(getAverage(leftDist,rightDist));
				}
				toAdd.add(newCluster);
			}
			counter++;
		}
		clusters.removeAll(toRemove);
		clusters.addAll(toAdd);
		return clusters;
	}

	private double getAverage(double distA, double distB){
		return (distA+distB)/2;
	}

	private double getMax(double distA, double distB){
		return Double.max(distA, distB);
	}

	private double getMin(double distA, double distB){
		return Double.min(distA, distB);
```

```
        }

        private Cluster getOtherChild(Cluster cluster, Cluster child){
                if(cluster.getleftChild().equals(child)){
                        return cluster.getrightChild();
                }
                return cluster.getleftChild();
        }

}
```

HierarchicalClustering.java

```
package hierarchicalClustering;

import java.util.ArrayList;

import fpgraphminer.FPGraph;
import model.Disease;

public class HierarchicalClustering {

        private ArrayList<Disease> graphDB;
        private String metric;

        public HierarchicalClustering(ArrayList<Disease> diseases, String metric){
                this.graphDB = diseases;
                this.metric = metric;
        }

        public ArrayList<String> getDiseaseNames(){
                ArrayList<String> names = new ArrayList<String>();
                for(Disease disease : this.graphDB){
                        names.add(disease.getTitle());
                }
                return names;
        }

        public double[][] getDistanceMatrix(){
                FPGraph fpg = new FPGraph(graphDB);
                ArrayList<String[]> fptable = fpg.makeDiseaseBitcodeTable(graphDB, fpg.
                     getDistinctEdges(graphDB),fpg.getDistinctDisconnectedNodes(graphDB));
                SimilarityMatrix sim = new SimilarityMatrix();
                if(this.metric.equals("Hamming")){
                        return sim.makeSimilarityMatrix(fptable, "Hamming");
                }
                return sim.makeSimilarityMatrix(fptable, "Jaccard");
        }



}
```

Dendrogram.java

```
package hierarchicalClustering;

import org.graphstream.graph.Graph;
import org.graphstream.graph.Node;
import org.graphstream.graph.implementations.SingleGraph;

public class Dendrogram {

        private Cluster root;
        private int width,height,maxLength;
        private int padding;

        public Dendrogram(Cluster root, int width, int height, int maxLen){
                this.root = root;
                this.width = width;
                this.height = height;
                this.maxLength = maxLen;
        }

        public void makeDendroRec(Graph graph, Node node, Cluster cluster, int xIncrement, int
             maxDist, int currentX, int currentY, int barlen){
                int xEndpoint = 0, clustDist=0;
                double parentDist = 0;
                if(!cluster.getChildList().isEmpty()){
                        parentDist = 0;
                        if(cluster.getParent()==null){
                                parentDist=0;
                        }else{
                                parentDist = cluster.getParent().getDistance();
                        }
                        clustDist = xIncrement*(int) (maxDist-cluster.getDistance());
                        xEndpoint = (int) (parentDist+clustDist);
                        int greaterX = 0;
                        if(currentX<=xEndpoint){
```

```
                                        Node horizontalEndPt = graph.addNode("H_endpt"+cluster.getName()+
                                             node.getId());
                                        horizontalEndPt.addAttribute("ui.label", String.format("%.2f",
                                             cluster.getDistance()));
                                        horizontalEndPt.addAttribute("layout.frozen");
                                        horizontalEndPt.addAttribute("x", xEndpoint);
                                        horizontalEndPt.addAttribute("y", currentY);
                                        horizontalEndPt.addAttribute("ui.style", "text-alignment: at-right
                                             ;");
                                        graph.addEdge(node.getId()+horizontalEndPt.getId(), node,
                                             horizontalEndPt);
                                        greaterX = xEndpoint;
                                }else{
                                        Node horizontalEndPt = graph.addNode("H_endpt"+cluster.getName()+
                                             node.getId());
                                        horizontalEndPt.addAttribute("ui.label", String.format("%.2f",
                                             cluster.getDistance()));
                                        horizontalEndPt.addAttribute("layout.frozen");
                                        horizontalEndPt.addAttribute("x", currentX);
                                        horizontalEndPt.addAttribute("y", currentY);
                                        horizontalEndPt.addAttribute("ui.style", "text-alignment: at-right
                                             ;");
                                        graph.addEdge(node.getId()+horizontalEndPt.getId(), node,
                                             horizontalEndPt);
                                        greaterX = currentX;
                                }
                                Node leftNodeBranch = graph.addNode("L_branch"+cluster.getName()+node.
                                     getId());
                                leftNodeBranch.addAttribute("layout.frozen");
                                leftNodeBranch.addAttribute("x", greaterX);
                                leftNodeBranch.addAttribute("y", currentY-(barlen/2)-12);
                                Node rightNodeBranch = graph.addNode("R_branch"+cluster.getName()+node.
                                     getId());
                                rightNodeBranch.addAttribute("layout.frozen");
                                rightNodeBranch.addAttribute("x", greaterX);
                                rightNodeBranch.addAttribute("y", currentY+(barlen/2)+12);
                                Node midPt = graph.getNode("H_endpt"+cluster.getName()+node.getId());
                                if(midPt!=null){
                                        graph.addEdge(leftNodeBranch.getId()+midPt.getId(), leftNodeBranch
                                             , midPt);
                                        graph.addEdge(rightNodeBranch.getId()+midPt.getId(), midPt,
                                             rightNodeBranch);
                                }
                                makeDendroRec(graph, leftNodeBranch ,cluster.getleftChild(), xIncrement,
                                     maxDist ,greaterX,currentY-(barlen/2)-12, barlen/2);
                                makeDendroRec(graph, rightNodeBranch ,cluster.getrightChild(), xIncrement,
                                      maxDist ,greaterX,currentY+(barlen/2)+12, barlen/2);
                        }else{
                                Node leaf = graph.addNode("Leaf "+cluster.getName());
                                leaf.addAttribute("layout.frozen");
                                double Xcoord = node.getNumber("x");
                                if((this.width-this.maxLength-this.padding)>Xcoord){
                                        leaf.addAttribute("xy", this.width-this.maxLength-this.padding,
                                             currentY);
                                }else{
                                        leaf.addAttribute("xy", Xcoord+xIncrement, currentY);
                                }
                                leaf.addAttribute("ui.label", cluster.getName());
                                leaf.addAttribute("ui.style", "text-size: 12; text-alignment: at-right;");
                                graph.addEdge(node.getId()+cluster.getName(), node, leaf);
                        }
        }

        public Graph makeDendrogram(){
                this.padding = 10;
                int maxDist = (int)(10*(Math.ceil(Math.abs(this.root.getDistance()/10))));
                int xIncrement = (this.width-this.maxLength-10)/maxDist;
                Graph graph = new SingleGraph("Dendrogram");
                Node root = graph.addNode(this.root.getName());
                root.addAttribute("layout.frozen");
                root.addAttribute("ui.style", "text-alignment: at-left;");
                root.addAttribute("x",this.padding);root.addAttribute("y",this.height/2);
                makeDendroRec(graph, root, this.root, xIncrement, maxDist, 10, this.height/2, this
                     .height/2);
                graph.addAttribute("ui.antialias");
                return graph;
        }

}
```

KGMLReader.java

```
package model;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class KGMLReader {
```

```
public Disease readXML(String filename){
        ArrayList<String> fileContents = new ArrayList<String>();
        try(BufferedReader br = new BufferedReader(new FileReader(filename))){
                String line;
                while((line = br.readLine()) != null) {
                                fileContents.add(line);
                }
        } catch (IOException e) {
                e.printStackTrace();
        }
        Disease disease = makeDisease(fileContents);
        return disease;
}

public Disease makeDisease(ArrayList<String> filecontents){
        String name = "";
        String title = "";
        ArrayList<Relation> Relations = new ArrayList<Relation>();
        ArrayList<Component> vertices = new ArrayList<Component>();
        boolean added = false;
        for(int i=0; i<filecontents.size(); i++){
                String currString = filecontents.get(i);
                if(currString.contains("<pathway")){
                        name = currString.substring(currString.indexOf("name=")+6,
                             currString.indexOf('"', currString.indexOf("name=")+6));
                }else if(currString.contains("<entry id=")){
                        Component v = makeComponent(currString);
                        if(!v.getName().equals(name)){
                                vertices.add(v);
                                added = true;
                        }else{
                                added = false;
                        }
                }else if(currString.contains("<relation entry1=")){
                        String type = currString.substring(currString.indexOf("type=")+6,
                             currString.indexOf('"',currString.indexOf("type=")+6));
                        Relation e = makeRelation(currString, vertices, type);
                        Relations.add(e);
                }else if(currString.contains("title=")){
                        title = currString.substring(currString.indexOf("title=")+7,
                             currString.indexOf('"',currString.indexOf("title=")+7));
                }else if(currString.contains("<subtype ")){
                        Relation e = Relations.get(Relations.size()-1);
                        e.addRelationSubtype(currString.substring(currString.indexOf("name
                             =")+6, currString.indexOf('"', currString.indexOf("name=")+6))
                             );
                }else if(currString.contains("<component id=")){
                        for(int compCtr=0; compCtr<vertices.size(); compCtr++){
                                if(currString.substring(currString.indexOf("<component id
                                     =")+15,currString.lastIndexOf('"')).equals(vertices.
                                     get(compCtr).getID())){
                                        vertices.get(vertices.size()-1).addGroupMember(
                                             vertices.get(compCtr));
                                }
                        }
                }else if(currString.contains("graphics name=")){
                        if(added){
                                if(currString.contains(",")){
                                        vertices.get(vertices.size()-1).setCommonName(
                                             currString.substring(currString.indexOf("name
                                             =")+6, currString.indexOf(',', currString.
                                             indexOf("name=")+6)));
                                }else{
                                        vertices.get(vertices.size()-1).setCommonName(
                                             currString.substring(currString.indexOf("name
                                             =")+6, currString.indexOf('"', currString.
                                             indexOf("name=")+6)));
                                }
                        }
                }
        }
        for(Component comp:vertices){
                if(comp.getType().equals("group")){
                        comp.setGroupCommonName();
                }
        }
        Disease disease = new Disease(name, title, Relations, vertices);
        return disease;
}
private Component makeComponent(String currString){
        String id = currString.substring(currString.indexOf("id=")+4, currString.indexOf
             ('"', currString.indexOf("id=")+4));
        String name = currString.substring(currString.indexOf("name=")+6, currString.
             indexOf('"', currString.indexOf("name=")+6));
        String type = currString.substring(currString.indexOf("type=")+6, currString.
             indexOf('"', currString.indexOf("type=")+6));
        Component v = new Component(id, name, type);
        return v;
}
private Relation makeRelation(String currString, ArrayList<Component> vertices, String
     type){
```

```
                String id1 = currString.substring(currString.indexOf("entry1=")+8, currString.
                     indexOf('"', currString.indexOf("entry1=")+8));
                String id2 = currString.substring(currString.indexOf("entry2=")+8, currString.
                     indexOf('"', currString.indexOf("entry2=")+8));
                ArrayList<String> relation = new ArrayList<String>();
                Component u = null, v=null;
                for(int componentCtr=0; componentCtr<vertices.size(); componentCtr++){
                        if(vertices.get(componentCtr).getID().equals(id1)){
                                u = vertices.get(componentCtr);
                        }else if(vertices.get(componentCtr).getID().equals(id2)){
                                v = vertices.get(componentCtr);
                        }
                }
                Relation e = new Relation(u,v,type,relation);
                return e;
        }
}
```

Component.java

```
package model;

import java.util.ArrayList;

public class Component {
        private String name;
        private String type;
        private String commonName;
        private String id;
        private ArrayList<Component> groupMembers;

        public Component(String id, String name, String type){
                this.id = id;
                this.type = type;
                if(!type.equals("group")){
                        this.name=name;
                }else{
                        this.name="group:";
                }
                this.groupMembers = new ArrayList<Component>();
        }

        public String getID(){
                return this.id;
        }

        public String getCommonName(){
                return this.commonName;
        }

        public String getName(){
                return this.name;
        }

        public String getType(){
                return this.type;
        }

        public ArrayList<Component> getGroupMembers(){
                return this.groupMembers;
        }

        public void addGroupMember(Component member){
                this.groupMembers.add(member);
                updateGroupName(member.getName());
        }

        public void setCommonName(String cName){
                this.commonName = cName;
        }

        public boolean isEqual(Component comp){
                if(this.name.equals(comp.getName()) && this.type.equals(comp.getType()) &&
                                this.id.equals(comp.getID())){
                        return true;
                }
                return false;
        }

        private void updateGroupName(String addedComponentName){
                if(this.groupMembers.isEmpty()){
                        this.name = this.name+" "+addedComponentName;
                }else{
                        this.name = this.name+"+"+addedComponentName;
                }
        }

        public void setGroupCommonName(){
                for(int memCtr=0; memCtr<this.groupMembers.size(); memCtr++){
                        if(memCtr==0){
```

```
                        this.commonName = this.groupMembers.get(memCtr).getCommonName();
                }else{
                        this.commonName = this.commonName+" "+this.groupMembers.get(memCtr
                            ).getCommonName();
                }
            }
        }
}
```

Relation.java

```
package model;

import java.util.ArrayList;

public class Relation {
        private Component component1;
        private Component component2;
        private ArrayList<String> subtype;
        private String type;
        private String name;
        private String commonName;

        public Relation(Component comp1, Component comp2, String type, ArrayList<String> subtype){
                this.component1 = comp1;
                this.component2 = comp2;
                this.type = type;
                this.subtype = subtype;
                this.name = comp1.getName()+"-"+this.type+"-"+comp2.getName();
                this.commonName = comp1.getCommonName()+"-"+this.type+"-"+comp2.getCommonName();
        }

        public ArrayList<Component> getComponents(){
                ArrayList<Component> components = new ArrayList<Component>();
                components.add(this.component1);
                components.add(this.component2);
                return components;
        }

        public ArrayList<String> getSubtype(){
                return this.subtype;
        }

        public void addRelationSubtype(String subtype){
                this.subtype.add(subtype);
        }

        public boolean isEqual(Relation rel){
                if(this.component1.isEqual(rel.getComponents().get(0)) &&
                                this.component2.isEqual(rel.getComponents().get(1)) &&
                                this.subtype.equals(rel.getSubtype())){
                        return true;
                }
                return false;
        }

        public String getName(){
                return this.name;
        }

        public String getCommonName(){
                return this.commonName;
        }

        public String getType(){
                return this.type;
        }
}
```

Disease.java

```
package model;

import java.util.ArrayList;
import java.util.Hashtable;

public class Disease {
        private ArrayList<Relation> relationList;
        private ArrayList<Component> componentList;
        private String name;
        private String title;

        public Disease(String name, String title, ArrayList<Relation> relations, ArrayList<
            Component> components){
                this.relationList = relations;
                this.componentList = components;
                this.name = name;
                this.title = title;
        }
        public void addRelation(Relation rel){
                this.relationList.add(rel);
        }

        public void setName(String name){
                this.name = name;
        }

        public void setTitle(String title){
                this.title = title;
        }

        public ArrayList<Relation> getRelations(){
                return this.relationList;
        }

        public String getName(){
                return this.name;
        }

        public String getTitle(){
                return this.title;
        }

        public ArrayList<Component> getComponents(){
                return this.componentList;
        }

        public ArrayList<String> getRelationNames(){
                ArrayList<String> relationNames = new ArrayList<String>();
                for(Relation relation : this.relationList){
                        relationNames.add(relation.getName());
                }
                return relationNames;
        }

        public ArrayList<Component> disconnectedComponents(){
                ArrayList<Component> dc = new ArrayList<Component>();
                ArrayList<String> connected = connectedComponents();
                for(Component comp : this.componentList){
                        if(!connected.contains(comp.getName())){
                                dc.add(comp);
                        }
                }
                return dc;
        }

        public ArrayList<String> disconnectedComponentNames(){
                ArrayList<String> dcNames = new ArrayList<String>();
                ArrayList<String> connected = connectedComponents();
                for(Component comp : this.componentList){
                        if(!connected.contains(comp.getName())){
                                dcNames.add(comp.getName());
                        }
                }
                return dcNames;
        }

        public Hashtable<String,String> getCommonNames(){
                Hashtable<String,String> nameEncoding = new Hashtable<String,String>();
                for(Component comp:this.componentList){
                        nameEncoding.put(comp.getName(), comp.getCommonName());
                }
                return nameEncoding;
        }

        private ArrayList<String> connectedComponents(){
                ArrayList<String> connected = new ArrayList<String>();
                for(Relation rel : this.relationList){
                        String comp1 = rel.getName().substring(0, rel.getName().indexOf('-'));
                        String comp2 = rel.getName().substring(rel.getName().lastIndexOf('-')+1,
                             rel.getName().length());
                        if(!connected.contains(comp1)){
                                connected.add(comp1);
                        }
                        if(!connected.contains(comp2)){
                                connected.add(comp2);
                        }
                }
                return connected;
        }

}
```

Database.java

```
package model;

import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Arrays;

public class Database {

        private Connection connect(){
                Connection conn = null;
                try{
                        File resourceFolder = new File("resources//db");

                        if(resourceFolder.exists()){
                                String url = "jdbc:sqlite:resources/db/diseaseDB.db";
                                conn = DriverManager.getConnection(url);
                        }else{
                                boolean resBool = new File("resources//db").mkdirs();
                                if(resBool){
                                        String url = "jdbc:sqlite:resources/db/diseaseDB.db";
                                        conn = DriverManager.getConnection(url);
                                        String sqlCreateDiseases = "CREATE TABLE IF NOT EXISTS
                                             Diseases (\n      diseaseTitle varchar(0,500) PRIMARY
                                             KEY,\n diseaseName varchar(0,50),\n relationList text
                                             ,\n componentList text);";
                                        Statement createDiseases = conn.createStatement();
                                        createDiseases.execute(sqlCreateDiseases);
                                        String sqlCreateRelations = "CREATE TABLE IF NOT EXISTS
                                             Relations (\n relationName varchar(0,500) PRIMARY KEY
                                             ,\n component1 varchar(0,500),\n component2 varchar
                                             (0,500),\n relationType varchar(0,50),\n
                                             relationSubtype text);";
                                        Statement createRelations = conn.createStatement();
                                        createRelations.execute(sqlCreateRelations);
                                        String sqlCreateComps = "CREATE TABLE IF NOT EXISTS
                                             Components (\n componentName varchar(0,500) PRIMARY
                                             KEY,\n componentType varchar(0,500),\n componentID
                                             integer,\n componentGroupMembers text,\n
                                             componentCommonName varchar(0,500));";
                                        Statement createComps = conn.createStatement();
                                        createComps.execute(sqlCreateComps);
                                }
                        }
                }catch(SQLException e){
                        System.out.println(e.getMessage());
                }
                return conn;
        }

        public ArrayList<String> getDiseasePathwayNames(){
                ArrayList<String> diseaseNames = new ArrayList<String>();
                String selectStmt = "SELECT diseaseName FROM Diseases";
                Connection conn = connect();
                try {
                        Statement connStmt = conn.createStatement();
                        ResultSet result = connStmt.executeQuery(selectStmt);
                        while(result.next()){
                                diseaseNames.add(result.getString("diseaseName"));
                        }
                } catch (SQLException e) {
                        System.out.println("Failed");
                }
                return diseaseNames;
        }

        public ArrayList<Disease> selectDisease(ArrayList<String> selectedDiseases){
                ArrayList<Disease> diseases = new ArrayList<Disease>();
                String selectStmt = "SELECT * FROM Diseases WHERE diseaseTitle in (";
                Connection conn = connect();
                for(int diseaseCtr=0; diseaseCtr<selectedDiseases.size()-1; diseaseCtr++){
                        selectStmt = selectStmt+"?,";
                }
                selectStmt = selectStmt+"?)";
                try {
                        PreparedStatement selectQuery = conn.prepareStatement(selectStmt);
                        for(int counter=0; counter<selectedDiseases.size(); counter++){
                                selectQuery.setString(counter+1, selectedDiseases.get(counter));
                        }
                        ResultSet result = selectQuery.executeQuery();
                        while(result.next()){
                                ArrayList<Relation> relations = makeRelations(result.getString("
                                     relationList"), conn);
                                ArrayList<Component> components = makeComponents(result.getString
                                     ("componentList"), conn);
                                Disease disease = new Disease(result.getString("diseaseName"),
                                     result.getString("diseaseTitle"),relations,components);
                                diseases.add(disease);
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return diseases;
        }

        private ArrayList<Component> makeComponents(String components, Connection conn){
                ArrayList<Component> componentList = new ArrayList<Component>();
                ArrayList<String> componentNames = new ArrayList<String>();
                String[] componentArr = components.split("\\|");
                for(String comp : componentArr){
                        Component component = makeComponent(comp,componentNames,componentList,
                             conn);
                        if(!componentNames.contains(comp)){
                                componentNames.add(comp);
                                componentList.add(component);
                        }
                }
                return componentList;
        }

        private ArrayList<Relation> makeRelations(String relationStr, Connection conn){
                ArrayList<Relation> relationList = new ArrayList<Relation>();
                ArrayList<Component> componentList = new ArrayList<Component>();
                ArrayList<String> componentNames = new ArrayList<String>();
                String[] relationArr = relationStr.split("\\|");
                for(String rel : relationArr){
                        String comp1 = rel.substring(0, rel.indexOf('-'));
                        String comp2 = rel.substring(rel.lastIndexOf('-')+1,rel.length());
                        String type = rel.substring(rel.indexOf('-')+1,rel.lastIndexOf('-'));
                        Component component1 = makeComponent(comp1, componentNames, componentList,
                             conn);
                        Component component2 = makeComponent(comp2, componentNames, componentList,
                             conn);
                        componentList.add(component1);componentNames.add(comp1);
                        componentList.add(component2);componentNames.add(comp2);
                        String selectStmt = "SELECT relationSubtype from Relations WHERE
                             relationName = ?";
                        try {
                                PreparedStatement selectQuery = conn.prepareStatement(selectStmt);
                                selectQuery.setString(1, rel);
                                ResultSet result = selectQuery.executeQuery();
                                ArrayList<String> subtype = new ArrayList<String>(Arrays.asList(
                                     result.getString("relationSubtype").split("\\|")));
                                Relation relation = new Relation(component1,component2,type,
                                     subtype);
                                relationList.add(relation);
                        } catch (SQLException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
                return relationList;
        }

        private Component makeComponent(String compName, ArrayList<String> componentNames,
             ArrayList<Component> components, Connection conn){
                String selectStmt = "SELECT * FROM Components WHERE componentName = ?";
                Component component = null;
                try {
                        PreparedStatement selectQuery = conn.prepareStatement(selectStmt);
                        selectQuery.setString(1, compName);
                        ResultSet result = selectQuery.executeQuery();
                        component = new Component(result.getString("componentID"),result.getString
                             ("componentName"),result.getString("componentType"));
                        component.setCommonName(result.getString("componentCommonName"));
                        if(result.getString("componentType").equals("group")){
                                String[] groupMemberNames = result.getString("
                                     componentGroupMembers").split("\\|");
                                for(String memberName : groupMemberNames){
                                        if(componentNames.contains(memberName)){
                                                component.addGroupMember(components.get(
                                                     componentNames.indexOf(memberName)));
                                        }else{
                                                Component member = makeComponent(memberName,
                                                     componentNames, components, conn);
                                                component.addGroupMember(member);
                                                components.add(member);
                                                componentNames.add(memberName);
                                        }
                                }
                        }
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return component;
        }

        public String[] getDiseaseNames(){
                ArrayList<String> diseaseNames = new ArrayList<String>();
                String selectStmt = "SELECT diseaseTitle FROM Diseases";
                Connection conn = connect();
                try {
                        Statement connStmt = conn.createStatement();
                        ResultSet result = connStmt.executeQuery(selectStmt);
                        while(result.next()){
                                diseaseNames.add(result.getString("diseaseTitle"));
                        }
                } catch (SQLException e) {
                        System.out.println("Failed");
                }
                String[] nameArr = new String[diseaseNames.size()];
                return diseaseNames.toArray(nameArr);
        }

        public String[][] getDiseaseNamesWithKEGGCode(){
                ArrayList<String[]> diseaseNames = new ArrayList<String[]>();
                String selectStmt = "SELECT diseaseTitle,diseaseName FROM Diseases";
                Connection conn = connect();
                try {
                        Statement connStmt = conn.createStatement();
                        ResultSet result = connStmt.executeQuery(selectStmt);
                        while(result.next()){
                                String codeName = result.getString("diseaseName");
                                String[] row = new String[2];
                                row[0] = result.getString("diseaseTitle");
                                row[1] = codeName.substring(codeName.indexOf(":")+1);
                                diseaseNames.add(row);
                        }
                } catch (SQLException e) {
                        System.out.println("Failed");
                }
                String[][] nameArr = new String[diseaseNames.size()][2];
                for(int ctr=0; ctr<diseaseNames.size(); ctr++){
                        nameArr[ctr] = diseaseNames.get(ctr);
                }
                return nameArr;
        }

        public void deleteDisease(ArrayList<String> deleteList) throws SQLException{
                String deleteStmt = "DELETE FROM Diseases WHERE diseaseTitle in (";
                Connection conn = connect();
                for(int diseaseCtr=0; diseaseCtr<deleteList.size()-1; diseaseCtr++){
                        deleteStmt = deleteStmt+"?,";
                }
                deleteStmt = deleteStmt+"?)";
                PreparedStatement deleteQuery = conn.prepareStatement(deleteStmt);
                for(int counter=0; counter<deleteList.size(); counter++){
                        deleteQuery.setString(counter+1, deleteList.get(counter));
                }
                deleteQuery.executeUpdate();
        }

        public void insertDisease(Disease disease){
                String insertStmt = "INSERT INTO Diseases(diseaseTitle,diseaseName,relationList,
                    componentList) VALUES(?,?,?,?)";
                Connection conn = connect();
                try {
                        PreparedStatement insertQuery = conn.prepareStatement(insertStmt);
                        insertQuery.setString(1, disease.getTitle());
                        insertQuery.setString(2, disease.getName());
                        insertQuery.setString(3, relationNames(disease));
                        insertQuery.setString(4, componentNames(disease));
                        insertQuery.executeUpdate();
                        for(Relation relation : disease.getRelations()){
                                insertRelation(relation, conn);
                        }
                        for(Component component : disease.getComponents()){
                                insertComponent(component, conn);
                        }
                } catch (SQLException e) {
                        System.out.println("Already Exists");
                }
        }

        public void insertComponent(Component component, Connection conn){
                String insertStmt = "INSERT INTO Components(componentName,componentType,
                     componentID,componentGroupMembers,componentCommonName) VALUES(?,?,?,?,?)";
                try {
                        PreparedStatement insertQuery = conn.prepareStatement(insertStmt);
                        insertQuery.setString(1, component.getName());
                        insertQuery.setString(2, component.getType());
                        insertQuery.setString(3, component.getID());
                        insertQuery.setString(4, groupMembers(component));
                        insertQuery.setString(5, component.getCommonName());
                        insertQuery.executeUpdate();
                        if(component.getType().equals("group")){
                                for(Component grpMember : component.getGroupMembers()){
                                        insertComponent(grpMember, conn);
                                }
                        }
                } catch (SQLException e) {
                        System.out.println("Already Exists");
                }
        }

        public void insertRelation(Relation relation, Connection conn){
                String insertStmt = "INSERT INTO Relations(relationName,component1,component2,
                    relationType,relationSubtype) VALUES(?,?,?,?,?)";
                try {
                        PreparedStatement insertQuery = conn.prepareStatement(insertStmt);
                        insertQuery.setString(1, relation.getName());
                        insertQuery.setString(2, relation.getComponents().get(0).getName());
                        insertQuery.setString(3, relation.getComponents().get(1).getName());
                        insertQuery.setString(4, relation.getType());
                        insertQuery.setString(5, subtypeConcat(relation.getSubtype()));
                        insertQuery.executeUpdate();
                } catch (SQLException e) {
                        System.out.println("Already Exists");
                }
        }

        private String relationNames(Disease disease){
                String concatenated = "";
                for(int relCtr=0; relCtr<disease.getRelationNames().size(); relCtr++){
                        if(relCtr==0){
                                concatenated = disease.getRelationNames().get(relCtr);
                        }else{
                                concatenated = concatenated+"|"+disease.getRelationNames().get(
                                    relCtr);
                        }
                }
                return concatenated;
        }

        private String componentNames(Disease disease){
                String concatenated = "";
                for(int compCtr=0; compCtr<disease.getComponents().size(); compCtr++){
                        if(compCtr==0){
                                concatenated = disease.getComponents().get(compCtr).getName();
                        }else{
                                concatenated = concatenated+"|"+disease.getComponents().get(
                                    compCtr).getName();
                        }
                }
                return concatenated;
        }

        private String groupMembers(Component component){
                String group = "";
                for(Component member : component.getGroupMembers()){
                        if(member.equals(component.getGroupMembers().get(0))){
                                group = member.getName();
                        }else{
                                group = group+"|"+member.getName();
                        }
                }
                return group;
        }

        private String subtypeConcat(ArrayList<String> relationList){
                String subtype = "";
                for(String sub : relationList){
                        if(sub.equals(relationList.get(0))){
                                subtype = sub;
                        }else{
                                subtype = subtype+"|"+sub;
                        }
                }
                return subtype;
        }
}
```

ReportMaker.java

```
package pdfGenerator;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.Date;

import com.itextpdf.text.BadElementException;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.FontFactory;
import com.itextpdf.text.Image;
import com.itextpdf.text.List;
import com.itextpdf.text.ListItem;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;

public class ReportMaker {

        private String filename;
        private ArrayList<String> imagePaths;
        private ArrayList<String> diseaseNames;
        private String functionality;
        private String username;

        public ReportMaker(String filename, ArrayList<String> imagePaths, String functionality){
                this.filename = filename;
                this.imagePaths = imagePaths;
                this.functionality = functionality;
                this.username = System.getProperty("user.name");
        }

        public void makeDendroReport(ArrayList<String> headers, double[][] data, String details)
             throws FileNotFoundException, DocumentException{
                Document report = new Document();
                PdfWriter.getInstance(report, new FileOutputStream(this.filename));
                report.open();
                initializeMetadata(report);
                addTitlePage(report, this.diseaseNames, details);
                addImages(report,null);
                addSIM(report,headers,data);
                report.close();
        }

        public void makeSearchReport(ArrayList<String> query) throws FileNotFoundException,
             DocumentException{
                Document report = new Document();
                PdfWriter.getInstance(report, new FileOutputStream(this.filename));
                report.open();
                initializeMetadata(report);
                addTitlePage(report,this.diseaseNames, buildQueryStr(query));
                addImages(report,null);
                report.close();
        }

        public void makeSubgraphReport(ArrayList<ArrayList<String>> edgeList, String details)
             throws FileNotFoundException, DocumentException{
                Document report = new Document();
                PdfWriter.getInstance(report, new FileOutputStream(this.filename));
                report.open();
                initializeMetadata(report);
                addTitlePage(report, this.diseaseNames, details);
                addImages(report, edgeList);
                report.close();
        }

        public void setDiseaseNames(ArrayList<String> names){
                this.diseaseNames = names;
        }

        private void initializeMetadata(Document pdf){
                pdf.addTitle("Disease Similarity Analysis Tool Report");
                pdf.addSubject("Analysis Report: "+this.functionality);
                pdf.addKeywords("Similarity Analysis, "+ this.functionality);
                pdf.addAuthor(this.username);
                pdf.addCreator(this.username);
        }

        private void addTitlePage(Document pdf, ArrayList<String> selectedDiseases, String
             subgraphDetails){
                Font largeFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 16, Font.NORMAL);
                Font mediumFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 14, Font.NORMAL);
                Font smallFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 12, Font.NORMAL);
                Paragraph preface = new Paragraph();
                try {
                        Image img = Image.getInstance(getClass().getResource("/images/softwarelogo
                             .PNG"));
                        float scaler = ((pdf.getPageSize().getWidth()-pdf.leftMargin()-pdf.
                             rightMargin()) / img.getWidth()) * 100;
                        img.scalePercent(scaler);
                        pdf.add(img);
                } catch (BadElementException | IOException e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                } catch (DocumentException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        addEmptyLine(preface, 2);
        preface.add(new Paragraph("Disease Similarity Analyzer Report",largeFont));

        addEmptyLine(preface, 1);
        preface.add(new Paragraph(this.functionality,mediumFont));
        addEmptyLine(preface, 1);
        preface.add(new Paragraph("Author: " + System.getProperty("user.name") + ", Date: " + new
             Date(),smallFont));


        Paragraph details = new Paragraph();
        addEmptyLine(details, 1);
        String diseases = "Analyzed Diseases:";
        List diseaseList = new List();
        for(String disease : selectedDiseases){
                diseaseList.add(new ListItem(disease));
        }
        details.add(new Paragraph(diseases));
        details.add(diseaseList);
        addEmptyLine(details, 1);
        if(!subgraphDetails.equals(null)){
                details.add(new Paragraph(subgraphDetails));
        }

        try {
                preface.setAlignment(Element.ALIGN_CENTER);
                        pdf.add(preface);
                        pdf.newPage();
                        pdf.add(details);
                } catch (DocumentException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        pdf.newPage();
        }

        private void addEmptyLine(Paragraph paragraph, int number) {
        for (int i = 0; i < number; i++) {
            paragraph.add(new Paragraph(" "));
        }
    }

        private void addImages(Document pdf, ArrayList<ArrayList<String>> edgeList){
                for(int imageCtr=0; imageCtr<imagePaths.size(); imageCtr++){
                        try {
                                Image img = Image.getInstance(imagePaths.get(imageCtr));
                                float scaler = ((pdf.getPageSize().getWidth()-pdf.leftMargin()-pdf
                                     .rightMargin()) / img.getWidth()) * 100;
                                img.scalePercent(scaler);
                                if(this.functionality.equals("Search Frequent Subgraphs")){
                                        pdf.add(new Paragraph("Subgraph "+(imageCtr+1)));
                                        pdf.add(img);
                                        addGraphEdges(pdf,edgeList.get(imageCtr));
                                }else if(this.functionality.equals("Search Diseases containing the
                                      query")){
                                        pdf.add(new Paragraph(diseaseNames.get(imageCtr)));
                                        pdf.add(img);
                                }else if(this.functionality.equals("Hierarchical Clustering")){
                                        pdf.add(new Paragraph("Dendrogram"));
                                        pdf.add(img);
                                }
                                pdf.newPage();
                        } catch (BadElementException | IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (DocumentException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }

        private void addGraphEdges(Document pdf, ArrayList<String> edges) throws DocumentException
             {
                Font mediumFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 14, Font.NORMAL);
                PdfPTable table = new PdfPTable(3);
                String[] headers = {"Component 1","Relation Type","Component 2"};

                Paragraph title = new Paragraph();
                title.add(new Paragraph("Relation List",mediumFont));
                addEmptyLine(title,1);

                for(String header:headers){
                        PdfPCell colHead = new PdfPCell(new Phrase(header));
                        colHead.setHorizontalAlignment(Element.ALIGN_CENTER);
                        table.addCell(colHead);
                }
                table.setHeaderRows(1);

                for(String edge : edges){
                        if(edge.indexOf('-') != edge.lastIndexOf('-')){
                                table.addCell(edge.substring(0, edge.indexOf('-')));
                                table.addCell(edge.substring(edge.indexOf('-')+1, edge.lastIndexOf
                                     ('-')));
                                table.addCell(edge.substring(edge.lastIndexOf('-')+1,edge.length()
                                     ));
```

```
                    }else{
                            table.addCell(edge);
                            table.addCell("n/a");
                            table.addCell("n/a");
                    }
            }

            pdf.add(title);
            pdf.add(table);
    }

    private void addSIM(Document pdf, ArrayList<String> headers, double[][] data) throws
         DocumentException{
            Font mediumFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 14, Font.NORMAL);
            Font smallFont = FontFactory.getFont(FontFactory.TIMES_ROMAN, 12, Font.NORMAL);
            NumberFormat format = new DecimalFormat("##.##");
            PdfPTable table = new PdfPTable(headers.size());

            Paragraph title = new Paragraph();
            title.add(new Paragraph("Similarity Matrix",mediumFont));
            addEmptyLine(title,1);

            for(String header : headers){
                    PdfPCell colHead = new PdfPCell(new Phrase(header,smallFont));
                    colHead.setHorizontalAlignment(Element.ALIGN_CENTER);
                    table.addCell(colHead);
            }
            table.setHeaderRows(1);

            for(int row=0; row<data.length; row++){
                    for(int col=0; col<data[row].length; col++){
                            table.addCell(new Phrase(format.format(data[row][col]),smallFont))
                                 ;
                    }
            }

            pdf.add(title);
            pdf.add(table);
    }

    private String buildQueryStr(ArrayList<String> queryList){
            String query = "Query: ";
            for(String str:queryList){
                    if(str.equals(queryList.get(queryList.size()-1))){
                            query = query+str;
                    }else{
                            query = query+str+", ";
                    }
            }
            return query;
    }
}
```

RestKEGG.java

```
package restKEGG;

import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;

import javax.imageio.ImageIO;

public class RestKEGG {

    public ArrayList<String[]> getPathwaysfromKEGG() throws IOException{
            URL url = null;
            BufferedReader in = null;
            String inputLine = "";
            ArrayList<String[]> result = new ArrayList<String[]>();
            try {
                    url = new URL("http://rest.kegg.jp/list/pathway/hsa");
                    in = new BufferedReader(new InputStreamReader(url.openStream()));
                    while ((inputLine = in.readLine()) != null){
                            String[] splitted = inputLine.split("\t");
                            splitted[0] = splitted[0].substring(splitted[0].indexOf(":")+1);
                            result.add(splitted);
                    }
                    in.close();
            }catch(MalformedURLException e) {
                    e.printStackTrace();
            }
            return result;
    }
```

```
        public ArrayList<String> readXMLfromKEGGURL(String url){
                URL keggURL = null;
                BufferedReader in = null;
                String inputLine = "";
                ArrayList<String> result = new ArrayList<String>();
                try {
                        keggURL = new URL(url);
                        in = new BufferedReader(new InputStreamReader(keggURL.openStream()));
                        while ((inputLine = in.readLine()) != null){
                                result.add(inputLine);
                        }
                        in.close();
                }catch(MalformedURLException e) {
                        e.printStackTrace();
                }catch(IOException e) {
                        System.out.println("NO INTERNET");
                        e.printStackTrace();
                }
                return result;
        }

        public void downloadImg(String url, String name){
                File dir = new File("resources//pathway images");
                if(!dir.exists()){
                        dir.mkdirs();
                }
        try {
            URL keggLink = new URL(url);
            BufferedImage image = ImageIO.read(keggLink);
            ImageIO.write(image, "png",new File("resources//pathway images//"+name+".png"));

        } catch (IOException e) {
                e.printStackTrace();
        }
        }
}
```

MainFrame.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

public class MainFrame{


        public static void main(String[] args){
                try {
                        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
                } catch (ClassNotFoundException | InstantiationException | IllegalAccessException
                                | UnsupportedLookAndFeelException e) {
                        e.printStackTrace();
                }
                System.setProperty("org.graphstream.ui.renderer", "org.graphstream.ui.j2dviewer.
                    J2DGraphRenderer");
                new MainFrame();
        }

        public MainFrame(){
                JFrame frame = new JFrame();
                Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
                int width = (int) (screenSize.getWidth()*0.75);
                int height =(int) (screenSize.getHeight()*0.9);
                frame.setTitle("Disease Similarity Analysis Tool");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setSize(width, height);
                frame.setPreferredSize(new Dimension(width, height));
                frame.setLocationRelativeTo(null);
                frame.setLayout(new BorderLayout());

                JPanel mainPanel = new MainPanel(width,height);

                frame.setContentPane(mainPanel);
                frame.setVisible(true);
        }
}
```

MainPanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class MainPanel extends JPanel{

        private static final long serialVersionUID = -7898061669240579696L;
        private int width,height;

        public MainPanel(int width, int height){
                this.width = width;
                this.height = height;
                initialize();
        }

        private void initialize(){
                setPreferredSize(new Dimension(width,height));
                setLayout(new BorderLayout());

                Font font = new Font("Arial",Font.PLAIN, 14);

                JPanel panel = new JPanel();
                panel.setOpaque(false);
                GridBagConstraints bagConstraints = new GridBagConstraints();
                bagConstraints.insets = new Insets(height/1000,width/100,0,width/100);
                panel.setLayout(new GridBagLayout());

                JPanel background = new BackgroundPanel(getClass().getResource("/images/
                    background2.png"));

                background.setLayout(new BorderLayout());

                JLabel logoLabel = new JLabel();
                try {
                        BufferedImage logo = ImageIO.read(getClass().getResource("/images/
                            softwarelogo.PNG"));
                        logoLabel.setIcon(new ImageIcon(logo));
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                bagConstraints.gridx = 0;
                bagConstraints.gridy = 0;
                bagConstraints.gridwidth = 3;
                bagConstraints.gridheight = 2;
                panel.add(logoLabel, bagConstraints);

                JButton startButton = new JButton("Start Analysis");
                startButton.setFont(font);
                startButton.addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent mousePress){
                                removeAll();
                            add(new DiseaseSelectionPanel(width,height), BorderLayout.CENTER);
                            revalidate();
                            repaint();
                        }
                });
                startButton.setPreferredSize(new Dimension(width/5,height/15));
                bagConstraints.gridx = 1;
                bagConstraints.gridy = 3;
                panel.add(startButton, bagConstraints);

                JButton dbButton = new JButton("Manage Diseases");
                dbButton.setFont(font);
                dbButton.addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent mousePress){
                                removeAll();
                            add(new DBPanel(width,height), BorderLayout.CENTER);
                            revalidate();
                            repaint();
                        }
                });
                dbButton.setPreferredSize(new Dimension(width/5,height/15));
                bagConstraints.gridx = 1;
                bagConstraints.gridy = 5;
                panel.add(dbButton, bagConstraints);

                JButton guideButton = new JButton("View Guide");
                guideButton.setFont(font);
                guideButton.addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent mousePress){
                                removeAll();
                            add(new UserGuidePanel(width,height), BorderLayout.CENTER);
                            revalidate();
                            repaint();
                        }
                });
                guideButton.setPreferredSize(new Dimension(width/5,height/15));
                bagConstraints.gridx = 1;
                bagConstraints.gridy = 7;
                panel.add(guideButton, bagConstraints);

                background.add(panel);

                add(background,BorderLayout.CENTER);

        }

}
```

BackgroundPanel.java

```
package view;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;
import java.net.URL;

import javax.swing.JPanel;

public class BackgroundPanel extends JPanel{

        /**
         *
         */
        private static final long serialVersionUID = 6100082185893042758L;
        private Image image;
        private int w, h;

        public BackgroundPanel(URL url){
                this.image = Toolkit.getDefaultToolkit().createImage(url);
                this.w = 0;
                this.h = 0;
        }

        public void setDimension(int w, int h){
                this.w = w;
                this.h = h;
        }

        @Override
        public void paintComponent(Graphics g){
                super.paintComponent(g);

                if(w==0 && h==0){
                        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
                        int width = (int) screenSize.getWidth();
                        int height = (int) screenSize.getHeight();
                        g.drawImage(image, 0, 0, width, height, this);
                }else{
                        g.drawImage(image, 0, 0, w, h, this);
                }
        }

}
```

DiseaseSelectionPanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Image;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.SwingUtilities;

import model.Database;
import model.Disease;
import restKEGG.RestKEGG;

public class DiseaseSelectionPanel extends JPanel{

        private static final long serialVersionUID = 5067452396077346505L;
        private JList<String> availableDiseases;
        private JPanel bg,bottom;
        private Image image;
        private JLabel imageLabel;
        private JButton confirm,back;
        private JScrollPane scrollAvailableDiseases,imageScroll;

        public DiseaseSelectionPanel(int w, int h){
                bg = new BackgroundPanel(getClass().getResource("/images/background2.png"));
                bg.setLayout(new BorderLayout(10,10));
                bg.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

                Database db = new Database();
                String[][] diseaseTuples = db.getDiseaseNamesWithKEGGCode();
                String[] diseaseList = getCommonNames(diseaseTuples);
                String[] diseaseKeggCode = getCodeNames(diseaseTuples);

                Image logo = null;
                try {
                        logo = ImageIO.read(getClass().getResource("/images/select.PNG")).
                             getScaledInstance(w/3, h/9, Image.SCALE_SMOOTH);
                } catch (IOException e) {
                        e.printStackTrace();
                }
                JLabel logoLabel = new JLabel(new ImageIcon(logo));
                logoLabel.setPreferredSize(new Dimension(w/3,h/9));
                bg.add(logoLabel, BorderLayout.PAGE_START);

                availableDiseases = new JList<String>(diseaseList);
                availableDiseases.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION)
                     ;
                availableDiseases.setFont(new Font("Arial",Font.PLAIN, 14));
                availableDiseases.addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent e){
                                if(SwingUtilities.isLeftMouseButton(e)){
                                        @SuppressWarnings("unchecked")
                                        JList<String> list = (JList<String>)e.getSource();
                                        int row = list.locationToIndex(e.getPoint());
                                        list.setSelectedIndex(row);
                                        String selected = diseaseKeggCode[row];
                                        try {
                                                File imagePath = new File("resources//pathway
                                                     images//"+selected+".png");
                                                if(!imagePath.exists()){
                                                        int answer = JOptionPane.
                                                             showConfirmDialog(null,"Pathway image
                                                              doesn't exist, would you like to
                                                             download the pathway image?","Image
                                                             not found",JOptionPane.YES_NO_OPTION)
                                                             ;
                                                        if(answer == JOptionPane.YES_OPTION){
                                                                RestKEGG rest = new RestKEGG();
                                                                rest.downloadImg("http://rest.
                                                                     kegg.jp/get/"+selected+"/
                                                                     image", selected);
                                                        }else{
                                                                imagePath = new File("/images/
                                                                     background2.png");
                                                        }
                                                }
                                                image = ImageIO.read(imagePath);
                                                JLabel imagePanel = new JLabel(new ImageIcon(image
                                                     ));
                                                imageScroll.setViewportView(imagePanel);
                                                revalidate();
                                                repaint();
                                        } catch (IOException e1) {
                                                e1.printStackTrace();
                                        }
                                }
                        }
                });
                scrollAvailableDiseases = new JScrollPane(availableDiseases);
                scrollAvailableDiseases.setPreferredSize(new Dimension(w/4,2*h/3));
                bg.add(scrollAvailableDiseases,BorderLayout.LINE_START);
                try {
                        image = ImageIO.read(getClass().getResource("/images/background.png"));
                        imageLabel = new JLabel(new ImageIcon(image));
                        imageScroll = new JScrollPane();
                        imageScroll.setViewportView(imageLabel);
                        imageScroll.setPreferredSize(new Dimension(2*w/3,2*h/3));
                        bg.add(imageScroll, BorderLayout.CENTER);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                bottom = new JPanel();
                bottom.setOpaque(false);
                confirm = new JButton("Finish Disease Selection");
                confirm.setFont(new Font("Arial",Font.PLAIN, 14));
                confirm.addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent mousePress){
                                ArrayList<String> selectedDiseases = null;
                                Database db = new Database();
                                try{
                                        selectedDiseases = (ArrayList<String>) availableDiseases.
                                            getSelectedValuesList();
                                        if(selectedDiseases.size()>=2){
                                                removeAll();
                                                ArrayList<Disease> diseases = db.selectDisease(
                                                    selectedDiseases);
                                            add(new DiseaseToolPanel(diseases,w,h), BorderLayout.
                                                CENTER);
                                            revalidate();
                                            repaint();
                                        }else{
                                                JOptionPane.showMessageDialog(null, "Please
                                                    selected at least two(2) diseases", "Error",
                                                    JOptionPane.ERROR_MESSAGE);
                                        }
                                }catch(ClassCastException ex){
                                        JOptionPane.showMessageDialog(null, "Empty selection", "
                                            Error", JOptionPane.ERROR_MESSAGE);
                                }
                        }
                });
                back = new JButton("Back to Main Menu");
                back.setFont(new Font("Arial",Font.PLAIN, 14));
                back.addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent mouse){
                                removeAll();
                            add(new MainPanel(w,h), BorderLayout.CENTER);
                            revalidate();
                            repaint();
                        }
                });
                bottom.add(confirm);
                bottom.add(back);
                bg.add(bottom,BorderLayout.PAGE_END);

                setLayout(new BorderLayout(10,10));
                add(bg);
        }

        private String[] getCodeNames(String[][] fromDB){
                String[] codeNames = new String[fromDB.length];
                for(int ctr=0; ctr<fromDB.length; ctr++){
                        codeNames[ctr] = fromDB[ctr][1];
                }
                return codeNames;
        }

        private String[] getCommonNames(String[][] fromDB){
                String[] commonNames = new String[fromDB.length];
                for(int ctr=0; ctr<fromDB.length; ctr++){
                        commonNames[ctr] = fromDB[ctr][0];
                }
                return commonNames;
        }
}
```

DiseaseToolPanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.Insets;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.IOException;
import java.util.ArrayList;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;

import model.Disease;

public class DiseaseToolPanel extends JPanel implements MouseListener{

        /**
         *
         */
        private static final long serialVersionUID = 2805713163150414821L;
        private JPanel bg, menu, bottom;
        private JButton dendroButton,queryButton,subgraphButton,backToSelection;
        private int width, height;
        private ArrayList<Disease> diseases;

        public DiseaseToolPanel(ArrayList<Disease> selectedDiseases, int width, int height){
                this.width = width;
                this.height = height;
                bg = new BackgroundPanel(getClass().getResource("/images/background2.png"));
                bg.setLayout(new BorderLayout());
                menu = new JPanel();
                menu.setLayout(new GridBagLayout());
                GridBagConstraints gbc = new GridBagConstraints();
                gbc.insets = new Insets(height/100,width/100,0,width/100);
                gbc.gridwidth = 3;
                gbc.gridheight = 2;
                diseases = selectedDiseases;

                Font font = new Font("Arial",Font.PLAIN, 14);

                Image logo = null;
                try {
                        logo = ImageIO.read(getClass().getResource("/images/task.PNG")).
                            getScaledInstance(width/3, height/8, Image.SCALE_SMOOTH);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                JLabel logoLabel = new JLabel(new ImageIcon(logo));
                logoLabel.setPreferredSize(new Dimension(width/3,height/8));
                gbc.gridx = 0;
                gbc.gridy = 0;
                gbc.gridwidth = 4;
                gbc.gridheight = 2;
                menu.add(logoLabel, gbc);

                gbc.gridx = 0;
                gbc.gridy = 3;
                dendroButton = new JButton("Hierarchical Clustering");
                dendroButton.setToolTipText("Perform Hierarchical clustering (Single, Average or
                    Complete Linkage)");
                dendroButton.setFont(font);
                dendroButton.setPreferredSize(new Dimension(width/3, height/15));
                dendroButton.addMouseListener(this);
                menu.add(dendroButton, gbc);
                gbc.gridx = 0;
                gbc.gridy = 5;
                queryButton = new JButton("Disease Search");
                queryButton.setFont(font);
                queryButton.setToolTipText("Search for diseases containing the selected relations
                    ");
                queryButton.setPreferredSize(new Dimension(width/3, height/15));
                queryButton.addMouseListener(this);
                menu.add(queryButton,gbc);
                gbc.gridx = 0;
                gbc.gridy = 7;
                subgraphButton = new JButton("Search Frequent Substructures");
                subgraphButton.setToolTipText("View the common frequent structures among the
                    diseases");
                subgraphButton.setFont(font);
                subgraphButton.setPreferredSize(new Dimension(width/3, height/15));
                subgraphButton.addMouseListener(this);
                menu.add(subgraphButton,gbc);
                menu.setOpaque(false);

                bottom = new JPanel();
                backToSelection = new JButton("Back to Disease Selection");
                backToSelection.addMouseListener(this);
                bottom.add(backToSelection);
                bottom.setOpaque(false);

                setLayout(new BorderLayout());
                bg.add(menu, BorderLayout.CENTER);
                bg.add(bottom, BorderLayout.PAGE_END);
                add(bg, BorderLayout.CENTER);
                addMouseListener(this);
        }

        @Override
        public void mouseClicked(MouseEvent arg0) {}
        @Override
        public void mouseEntered(MouseEvent e) {}

        @Override
        public void mouseExited(MouseEvent e) {}

        @Override
        public void mousePressed(MouseEvent e) {
                if(e.getComponent().equals(dendroButton)){
                        removeAll();
                    add(new DendrogramPanel(diseases, width, height), BorderLayout.CENTER);
                    revalidate();
                    repaint();
                }
                if(e.getComponent().equals(queryButton)){
                        removeAll();
                    add(new GraphSearchPanel(diseases,width, height), BorderLayout.CENTER);
                    revalidate();
                    repaint();
                }
                if(e.getComponent().equals(subgraphButton)){
                        removeAll();
                    add(new SubgraphPanel(diseases,width,height), BorderLayout.CENTER);
                    revalidate();
                    repaint();
                }
                if(e.getComponent().equals(backToSelection)){
                        removeAll();
                    add(new DiseaseSelectionPanel(width,height), BorderLayout.CENTER);
                    revalidate();
                    repaint();
                }
        }

        @Override
        public void mouseReleased(MouseEvent e) {}

}
```

DendrogramPanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics2D;
import java.awt.GraphicsEnvironment;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Date;

import javax.imageio.ImageIO;
import javax.swing.AbstractButton;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTable;
import javax.swing.SwingConstants;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

import org.graphstream.graph.Graph;
import org.graphstream.ui.swingViewer.ViewPanel;
import org.graphstream.ui.view.Viewer;

import com.itextpdf.text.DocumentException;

import hierarchicalClustering.AverageLinkage;
import hierarchicalClustering.Cluster;
import hierarchicalClustering.Dendrogram;
import hierarchicalClustering.HierarchicalClustering;
import model.Disease;
import pdfGenerator.ReportMaker;

public class DendrogramPanel extends JPanel implements MouseListener{

        private static final long serialVersionUID = -8614974066473065592L;
        private JPanel bg, options, output, top, bottom, outputCont, tablePanel;
        private JButton back, export, computeHierarchy;
        private JRadioButton jsi, hd, min, max, ave;
        private ButtonGroup btngrp, linkgrp;
        private Image panLogo,ZupLogo,ZdownLogo;
        private ArrayList<Disease> diseases;
        private int width, height;
        private Graph graph;
        private double[][] distances;
        private ArrayList<String> names;

        public DendrogramPanel(ArrayList<Disease> diseases, int width, int height){
                makePanel(diseases, width, height);
        }

        private void makePanel(ArrayList<Disease> diseases, int width, int height){
                this.width = width;
                this.height = height;
                this.diseases = diseases;
                setLayout(new BorderLayout());

                setTopPanel();
                setBG();
                setOptionsPanel();
                setOutputPanel();
                setBottomPanel();

                bg.add(top,BorderLayout.PAGE_START);
                bg.add(options,BorderLayout.LINE_START);
                bg.add(outputCont,BorderLayout.CENTER);
                bg.add(bottom, BorderLayout.PAGE_END);
                add(bg, BorderLayout.CENTER);
                addMouseListener(this);
        }

        private void setTopPanel(){
                top = new JPanel();
                top.setLayout(new BorderLayout());
                back = new JButton("Back");
                back.addMouseListener(this);
                top.add(back, BorderLayout.LINE_START);
                top.setOpaque(false);
        }

        private void setBG(){

                bg = new BackgroundPanel(getClass().getResource("/images/background.png"));
                bg.setLayout(new BorderLayout(10,10));
                bg.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        }

        private void setOptionsPanel(){
                options = new JPanel(new BorderLayout(10,10));
                options.setOpaque(false);
                options.setPreferredSize(new Dimension(width/3, height));

                TitledBorder paramBorder = BorderFactory.createTitledBorder(BorderFactory.
                    createEtchedBorder(EtchedBorder.LOWERED), "Metric");
                paramBorder.setTitleFont(new Font("Arial",Font.PLAIN, 14));
                paramBorder.setTitleColor(Color.WHITE);

                JPanel paramPanel = new JPanel(new BorderLayout());
                paramPanel.setPreferredSize(new Dimension(width/4,height/2));
                paramPanel.setOpaque(false);

                JPanel parameters = new JPanel();
                parameters = new JPanel();
                parameters.setLayout(new GridBagLayout());
                GridBagConstraints gbc = new GridBagConstraints();
                parameters.setPreferredSize(new Dimension(width/4,height/2));
                gbc.anchor = GridBagConstraints.LINE_END;
                gbc.gridx = 0;
                gbc.gridy = 0;
                parameters.add(new JLabel("Metric: "), gbc);
                gbc.gridy+=2;
                parameters.add(new JLabel("Linkage: "), gbc);

                gbc.anchor = GridBagConstraints.LINE_START;
                jsi = new JRadioButton("Jaccard Similarity Index");
                jsi.setOpaque(false);
                jsi.setSelected(true);
                hd = new JRadioButton("Hamming Distance");
                hd.setOpaque(false);
                btngrp = new ButtonGroup();
                btngrp.add(jsi);
                btngrp.add(hd);
                gbc.gridx = 1;
                gbc.gridy = 0;
                parameters.add(jsi , gbc);
                gbc.gridy++;
                parameters.add(hd, gbc);
                min = new JRadioButton("Single Linkage");
                min.setOpaque(false);
                max = new JRadioButton("Complete Linkage");
                max.setOpaque(false);
                ave = new JRadioButton("Average Linkage");
                ave.setOpaque(false);
                ave.setSelected(true);
                linkgrp = new ButtonGroup();
                linkgrp.add(min);
                linkgrp.add(ave);
                linkgrp.add(max);
                gbc.gridy++;
                parameters.add(min,gbc);
                gbc.gridy++;
                parameters.add(ave,gbc);
                gbc.gridy++;
                parameters.add(max,gbc);
                computeHierarchy = new JButton("Compute Hierarchy");
                gbc.gridx = 1;
                gbc.gridy+=2;
                computeHierarchy.addMouseListener(this);
                parameters.add(computeHierarchy,gbc);
                parameters.setBackground(Color.WHITE);
                paramPanel.add(parameters, BorderLayout.CENTER);
                paramPanel.setBorder(paramBorder);

                JTabbedPane tabbedPane = new JTabbedPane();
                tabbedPane.setPreferredSize(new Dimension(width/3, height/2));
                tabbedPane.setOpaque(false);

                JPanel controls = new JPanel(new GridBagLayout());
                controls.setBackground(Color.WHITE);
                setControlImages();
                JLabel panLabel = new JLabel(new ImageIcon(panLogo));
                JLabel upLabel = new JLabel(new ImageIcon(ZupLogo));
                JLabel downLabel = new JLabel(new ImageIcon(ZdownLogo));
                controls.setPreferredSize(new Dimension(width/4, height/2));
                gbc.anchor = GridBagConstraints.LINE_START;
                gbc.gridx = 0;
                gbc.gridy = 0;
                controls.add(new JLabel("Panning:"), gbc);
                gbc.gridx = 1;
                gbc.gridy++;
                controls.add(panLabel, gbc);
                gbc.gridx = 0;
                gbc.gridy++;
                controls.add(new JLabel("Zoom in: "), gbc);
                gbc.gridx = 1;
                gbc.gridy++;
                controls.add(upLabel, gbc);
                gbc.gridx = 0;
                gbc.gridy++;
                controls.add(new JLabel("Zoom out: "), gbc);
                gbc.gridx = 1;
                gbc.gridy++;
                controls.add(downLabel, gbc);

                tablePanel = new JPanel(new BorderLayout());
                tablePanel.setPreferredSize(new Dimension(width/4, height/2));

                TitledBorder contBorder = BorderFactory.createTitledBorder(BorderFactory.
                    createEtchedBorder(EtchedBorder.LOWERED), "Matrix and Controls");
                contBorder.setTitleFont(new Font("Arial",Font.PLAIN, 14));
                contBorder.setTitleJustification(TitledBorder.LEFT);
                contBorder.setTitleColor(Color.WHITE);

                tabbedPane.setBorder(contBorder);
                tabbedPane.addTab("Matrix", tablePanel);
                tabbedPane.addTab("Controls", controls);

                options.add(paramPanel, BorderLayout.CENTER);
                options.add(tabbedPane, BorderLayout.PAGE_END);
        }

        private void setControlImages(){
                panLogo = null;ZupLogo = null;ZdownLogo = null;
                try {
                        panLogo = ImageIO.read(getClass().getResource("/images/arrowkeys.png")).
                            getScaledInstance(width/10, height/6, Image.SCALE_SMOOTH);
                        ZupLogo = ImageIO.read(getClass().getResource("/images/page-up.png")).
                            getScaledInstance(width/15, height/11, Image.SCALE_SMOOTH);
                        ZdownLogo = ImageIO.read(getClass().getResource("/images/page-down.png")).
                            getScaledInstance(width/15, height/11, Image.SCALE_SMOOTH);
                } catch (IOException e) {
```

```
                    System.out.println("IMAGE IO ERROR");
                }
        }

        private void setOutputPanel(){
                outputCont = new JPanel(new BorderLayout(10,10));
                outputCont.setOpaque(false);

                JLabel outputLabel = new JLabel("Dendrogram", SwingConstants.CENTER);
                outputLabel.setForeground(Color.WHITE);
                outputLabel.setFont(new Font("Arial",Font.PLAIN, 14));

                output = new JPanel();
                output.setLayout(new BorderLayout());
                output.setPreferredSize(new Dimension(2*width/3, height));
                output.setBackground(Color.WHITE);

                outputCont.add(outputLabel, BorderLayout.PAGE_START);
                outputCont.add(output,BorderLayout.CENTER);
        }

        private void setBottomPanel(){
                bottom = new JPanel();
                export = new JButton("Export results as PDF");
                export.addMouseListener(this);
                bottom.add(export);
                bottom.setOpaque(false);
        }

         private String getSelectedRadioButton(ButtonGroup btnGrp, String type) {
                ArrayList<AbstractButton> buttons = Collections.list(btnGrp.getElements());
                String selected = "";
        for (AbstractButton button : buttons){
                if(type.equals("Metric")){
                    if(button.isSelected()){
                        if(button.equals(hd)){
                                selected = "Hamming";
                        }else{
                                selected = "Jaccard";
                        }
                    }
                }else if(type.equals("Linkage")){
                        if(button.isSelected()){
                                if(button.equals(min)){
                                        selected = "min";
                                }else if(button.equals(max)){
                                        selected = "max";
                                }else{
                                        selected = "ave";
                                }
                        }
                }
        }
        return selected;
    }

        @Override
        public void mouseClicked(MouseEvent arg0) {}

        @Override
        public void mouseEntered(MouseEvent arg0) {}

        @Override
        public void mouseExited(MouseEvent arg0) {}

        @Override
        public void mousePressed(MouseEvent arg0) {}

        @Override
        public void mouseReleased(MouseEvent e) {
                if(e.getComponent().equals(back)){
                        removeAll();
                    add(new DiseaseToolPanel(diseases,width,height), BorderLayout.CENTER);
                    revalidate();
                    repaint();
                }
                if(e.getComponent().equals(export)){
                        BufferedImage bufferedImage = createImage(output);
                        File dir = new File("resources//exported images");
                        File outputfile = null;
                        if(dir.exists()){
                                outputfile = new File("resources//exported images//Dendrogram.png
                                    ");
                        }else{
                                boolean dirBool = new File("resources//exported images").mkdir();
                                if(dirBool){
                                        outputfile = new File("resources//exported images//
                                            Dendrogram.png");
                                }
                        }
                try {
                                ImageIO.write(bufferedImage, "png", outputfile);
                    String date = new SimpleDateFormat("MM-dd-yyyy HH,mm,ss").format(
                         new Date());
                    String filename = "[DSA] Hierarchical Clustering "+date+".pdf";
                    ArrayList<String> imagePaths = new ArrayList<String>();
                    imagePaths.add("./resources//exported images//Dendrogram.png");
                    ReportMaker reportGen = new ReportMaker(filename, imagePaths, "
                         Hierarchical Clustering");
                    try {
                            if(names!=null && distances!=null){
                                    reportGen.setDiseaseNames(names);
                                    String[] details = getStringDetails();
                                    reportGen.makeDendroReport(names,distances,"Metric
                                         : "+details[0]+"\nLinkage: "+details[1]);
                                    JOptionPane.showMessageDialog(null, "Report
                                         generated", "Success!", JOptionPane.
                                         INFORMATION_MESSAGE);
                            }else{
                                    JOptionPane.showMessageDialog(null, "Please
                                         execute the clustering before exporting", "
                                         Error", JOptionPane.ERROR_MESSAGE);
                            }
                    } catch (FileNotFoundException e1) {
                            JOptionPane.showMessageDialog(null, "Document Error", "
                                 Error", JOptionPane.ERROR_MESSAGE);
                    } catch (DocumentException e1) {
                            JOptionPane.showMessageDialog(null, "Document Error", "
                                 Error", JOptionPane.ERROR_MESSAGE);
                    } catch (NullPointerException e1){
                            JOptionPane.showMessageDialog(null, "Please execute the
                                 clustering before exporting", "Error", JOptionPane.
                                 ERROR_MESSAGE);
                            e1.printStackTrace();
                    }
        } catch (IOException img) {
                    JOptionPane.showMessageDialog(null, "Image export Error", "Error",
                          JOptionPane.ERROR_MESSAGE);
                    img.printStackTrace();
            }

        }
        if(e.getComponent().equals(computeHierarchy)){
                String metric = getSelectedRadioButton(btngrp,"Metric");
                String linkage = getSelectedRadioButton(linkgrp,"Linkage");
                HierarchicalClustering hc = new HierarchicalClustering(this.diseases,
                     metric);
                this.distances = hc.getDistanceMatrix();
                this.names = hc.getDiseaseNames();
                tablePanel.removeAll();
                JTable table = makeMatrix(names, distances);
                table.setPreferredSize(new Dimension(width/4, height/2));
                JScrollPane scrollable = new JScrollPane(table,JScrollPane.
                     VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.
                     HORIZONTAL_SCROLLBAR_AS_NEEDED);
                scrollable.setPreferredSize(new Dimension(width/4, height/2));
                tablePanel.add(scrollable, BorderLayout.CENTER);
                AverageLinkage averageLink = new AverageLinkage();
                Cluster root = averageLink.cluster(distances, names, linkage);
                output.removeAll();
                int maxLen = maxNameLength();
                Dendrogram gd = new Dendrogram(root, output.getWidth(), output.getHeight()
                     ,maxLen);
                graph = gd.makeDendrogram();
                Viewer viewer = new Viewer(graph, Viewer.ThreadingModel.
                     GRAPH_IN_ANOTHER_THREAD);
            ViewPanel viewPanel = viewer.addDefaultView(false);
            viewPanel.setPreferredSize(output.getPreferredSize());
            viewPanel.setBackground(Color.WHITE);
            output.add(viewPanel, BorderLayout.CENTER);
                 revalidate();
            repaint();
        }
}

private int maxNameLength(){
        int max = 0;
        FontMetrics fm = null;
        if(fontChecker(Font.SANS_SERIF)){
                fm = this.getFontMetrics(new Font(Font.SANS_SERIF, Font.PLAIN, 16));
        }else{
                fm = this.getFontMetrics(this.getFont());
        }
        for(String name:this.names){
                if(fm.stringWidth(name)>max){
                        max = fm.stringWidth(name);
                }
        }
        return max;
}

private boolean fontChecker(String fontname){
        boolean exists = false;
        GraphicsEnvironment g = GraphicsEnvironment.getLocalGraphicsEnvironment();
```

```
                ArrayList<String> fonts = new ArrayList<String>(Arrays.asList(g.
                    getAvailableFontFamilyNames()));
                if(fonts.contains(fontname)){
                        exists = true;
                }
                return exists;
        }

        private JTable makeMatrix(ArrayList<String> names, double[][] data){
                String[] nameArr = names.toArray(new String[names.size()]);
                String[][] strData = new String[data.length][data.length];
                for(int rowCtr=0; rowCtr<data.length; rowCtr++){
                        for(int colCtr=0; colCtr<data[rowCtr].length; colCtr++){
                                strData[rowCtr][colCtr] = String.format("%.2f",data[rowCtr][colCtr
                                    ]);
                        }
                }
                JTable table = new JTable(strData,nameArr){
                        private static final long serialVersionUID = 5139240796521787980L;
                        public boolean isCellEditable(int row, int column) {
                return false;
                        };
                };
                return table;
        }

        private String[] getStringDetails(){
                String[] details = new String[2];
                if(getSelectedRadioButton(btngrp,"Metric").equals("Hamming")){
                        details[0] = "Hamming Distance";
                }else{
                        details[0] = "Jaccard Similarity Index";
                }
                if(getSelectedRadioButton(linkgrp,"Linkage").equals("min")){
                        details[1] = "Single Linkage";
                }else if(getSelectedRadioButton(linkgrp,"Linkage").equals("max")){
                        details[1] = "Complete Linkage";
                }else{
                        details[1] = "Average Linkage";
                }
                return details;
        }

        public BufferedImage createImage(JPanel panel) {

        int w = panel.getWidth();
        int h = panel.getHeight();
        BufferedImage bi = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
        Graphics2D g = bi.createGraphics();
        panel.print(g);
        return bi;
    }
}
```

GraphSearchPanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.SwingConstants;

import org.graphstream.graph.Graph;
import org.graphstream.stream.file.FileSinkImages;
import org.graphstream.stream.file.FileSinkImages.LayoutPolicy;
import org.graphstream.stream.file.FileSinkImages.OutputType;
import org.graphstream.stream.file.FileSinkImages.Resolutions;
import org.graphstream.ui.swingViewer.ViewPanel;
import org.graphstream.ui.view.Viewer;

import com.itextpdf.text.DocumentException;
import fpgraphminer.FPGfunctions;
import fpgraphminer.FPGraph;
import model.Disease;
import pdfGenerator.ReportMaker;

public class GraphSearchPanel extends JPanel implements MouseListener{

        private static final long serialVersionUID = 8854793388938416071L;
        private JPanel bg, bottom, top, options, graphPanel, output;
        private JLabel outputLabel;
        private JButton back, export, search, fp;
        private JCheckBox signif;
        private int width, height, counter;
        private ArrayList<Disease> graphDB;
        private FPGfunctions fpg;
        private FPGraph fpgraph;
        private JList<String> availableEdges;
        private ArrayList<String> query;
        private ArrayList<String> keggEdgeCodename,distinctElements;

        public GraphSearchPanel(ArrayList<Disease> selectedDiseases, int width, int height){
                this.width = width;
                this.height = height;
                this.graphDB = selectedDiseases;
                this.counter = 0;
                this.fpgraph = new FPGraph(this.graphDB);
                this.fpg = new FPGfunctions(this.graphDB);

                makePanel();
        }

        private void makePanel(){
                setLayout(new BorderLayout());

                setBG();
                setTop();
                setOptions();
                setOutput();
                setBottom();

                bg.add(top, BorderLayout.PAGE_START);
                bg.add(options, BorderLayout.LINE_START);
                bg.add(output, BorderLayout.CENTER);
                bg.add(bottom, BorderLayout.PAGE_END);
                add(bg, BorderLayout.CENTER);
                addMouseListener(this);
        }

        private void setBG(){
                bg = new BackgroundPanel(getClass().getResource("/images/background.png"));
                bg.setLayout(new BorderLayout(10,10));
                bg.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        }

        private void setTop(){
                top = new JPanel();
                top.setLayout(new BorderLayout());
                top.setOpaque(false);
                back = new JButton("Back");
                back.addMouseListener(this);
                top.add(back, BorderLayout.LINE_START);
        }

        private void setOptions(){
                options = new JPanel(new BorderLayout(10,10));
                options.setPreferredSize(new Dimension(width/3, height));
                options.setOpaque(false);

                JLabel optionLabel = new JLabel("Select relations", SwingConstants.CENTER);
                optionLabel.setForeground(Color.WHITE);
                optionLabel.setFont(new Font("Arial",Font.PLAIN, 14));
                options.add(optionLabel, BorderLayout.PAGE_START);

                keggEdgeCodename = new ArrayList<String>();
                keggEdgeCodename.addAll(this.fpgraph.getDistinctEdges(this.graphDB));
                keggEdgeCodename.addAll(this.fpgraph.getDistinctDisconnectedNodes(this.graphDB));
                distinctElements = new ArrayList<String>();
                distinctElements.addAll(this.fpgraph.getDistinctEdgesCommonName());
                distinctElements.addAll(this.fpgraph.getDCCommonNames());
                String[] distinctEdges = distinctElements.toArray(new String[distinctElements.size
                    ()]);
                availableEdges = new JList<String>(distinctEdges);
                availableEdges.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
                availableEdges.setFont(new Font("Arial",Font.PLAIN, 14));
                JScrollPane scrollable = new JScrollPane(availableEdges);
                options.add(scrollable,BorderLayout.CENTER);

                JPanel bottom = new JPanel(new BorderLayout());
                bottom.setOpaque(false);
                search = new JButton("Search");
                search.addMouseListener(this);
                signif = new JCheckBox("Show component significance");
                signif.setOpaque(false);
                signif.setForeground(Color.WHITE);
                bottom.add(signif,BorderLayout.PAGE_START);
                bottom.add(search,BorderLayout.PAGE_END);

                options.add(bottom, BorderLayout.PAGE_END);
        }

        private void setOutput(){
                output = new JPanel(new BorderLayout(10,10));
                output.setOpaque(false);

                outputLabel = new JLabel("Disease: ", SwingConstants.CENTER);
                outputLabel.setForeground(Color.WHITE);
                outputLabel.setFont(new Font("Arial",Font.PLAIN, 14));

                graphPanel = new JPanel(new BorderLayout());

                output.add(outputLabel, BorderLayout.PAGE_START);
                output.add(graphPanel, BorderLayout.CENTER);
        }

        private void setBottom(){
                bottom = new JPanel();
                export = new JButton("Export results as PDF");
                export.addMouseListener(this);
                fp = new JButton("View FP Table & Graph");
                fp.addMouseListener(this);
                bottom.add(export);
                bottom.add(fp);
                bottom.setOpaque(false);
        }

        @Override
        public void mouseClicked(MouseEvent arg0) {}

        @Override
        public void mouseEntered(MouseEvent arg0) {}

        @Override
        public void mouseExited(MouseEvent arg0) {}

        @Override
        public void mousePressed(MouseEvent arg0) {}

        @Override
        public void mouseReleased(MouseEvent e) {
                if(e.getComponent().equals(back)){
                        removeAll();
                    add(new DiseaseToolPanel(graphDB,width,height), BorderLayout.CENTER);
                    revalidate();
                    repaint();
                }
                if(e.getComponent().equals(search)){
                        try{
                                int[] indexes = availableEdges.getSelectedIndices();
                                this.query = new ArrayList<String>();
                                for(int index : indexes){
                                        this.query.add(this.keggEdgeCodename.get(index));
                                }
                                if(!query.isEmpty()){
                                        ArrayList<Disease> diseases = fpg.getGraphsContainingQuery
                                             (query);
                                        int diseaseCount = diseases.size();
                                        Graph graph;
                                        if(diseaseCount!=0){
                                                if(counter>=diseaseCount){
                                                        counter = 0;
                                                        graphPanel.removeAll();
                                                        graph = fpg.showQueryGraph(diseases.get(
                                                             counter), query);
                                                }else{
                                                        graphPanel.removeAll();
                                                        graph = fpg.showQueryGraph(diseases.get(
                                                             counter), query);
                                                }
                                                Viewer viewer = new Viewer(graph, Viewer.
                                                     ThreadingModel.GRAPH_IN_ANOTHER_THREAD);
                                            ViewPanel viewPanel = viewer.addDefaultView(false);
                                            viewer.enableAutoLayout();
                                            outputLabel.setText("Disease "+(counter+1)+" of "+
                                                 diseaseCount+" :\t"+diseases.get(counter).getTitle
                                                 ());
                                            graphPanel.add(viewPanel, BorderLayout.CENTER);
                                            revalidate();
                                            repaint();
                                            counter++;
                                        }
                                }else{
                                        JOptionPane.showMessageDialog(null, "Please select at
                                             least one relation", "Error", JOptionPane.
```

```
                                                ERROR_MESSAGE);
                                }
                        }catch(ClassCastException e1){
                                JOptionPane.showMessageDialog(null, "No diseases containing the
                                        query was found", "Empty output", JOptionPane.
                                        INFORMATION_MESSAGE);
                        }
                }
                if(e.getComponent().equals(export)){
                        String date = new SimpleDateFormat("MM-dd-yyyy HH,mm,ss").format(new Date
                                ());
                        String filename = "[DSA] Disease Search "+date+".pdf";
                        ArrayList<Disease> diseases = new ArrayList<Disease>();
                        try{
                                diseases = fpg.getGraphsContainingQuery(this.query);
                        }catch(NullPointerException e1){
                                JOptionPane.showMessageDialog(null, "No output can be exported", "
                                        Error", JOptionPane.ERROR_MESSAGE);
                        }
                        ArrayList<String> imagePaths = new ArrayList<String>();
                        if(!diseases.isEmpty()){
                                ArrayList<Graph> graphs = fpg.getDiseasesGraphStream(query,
                                        diseases);
                                ArrayList<String> names = new ArrayList<String>();
                                for(int graphCtr=0; graphCtr<graphs.size(); graphCtr++){
                                        names.add(diseases.get(graphCtr).getTitle());
                                        FileSinkImages pic = new FileSinkImages(OutputType.PNG,
                                                Resolutions.VGA);
                                        pic.setLayoutPolicy(LayoutPolicy.
                                                COMPUTED_FULLY_AT_NEW_IMAGE);
                                        try {
                                                pic.writeAll(graphs.get(graphCtr), "resources//
                                                        exported images//Searched Disease "+graphCtr
                                                        +".png");
                                                imagePaths.add("./resources//exported images//
                                                        Searched Disease "+graphCtr+".png");
                                        } catch (IOException e1) {
                                                System.out.println("Export failed");
                                        }
                                }
                                ReportMaker reportGen = new ReportMaker(filename, imagePaths, "
                                        Search Diseases containing the query");
                                try {
                                        reportGen.setDiseaseNames(names);
                                        reportGen.makeSearchReport(this.query);
                                        JOptionPane.showMessageDialog(null, "Report generated", "
                                                Success!", JOptionPane.INFORMATION_MESSAGE);
                                } catch (FileNotFoundException | DocumentException e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                }
                        }else{
                                JOptionPane.showMessageDialog(null, "No diseases containing the
                                        query was found", "Empty output", JOptionPane.
                                        INFORMATION_MESSAGE);
                        }
                }
                if(e.getComponent().equals(fp)){
                        removeAll();
                    add(new FPpanel(width,height,this.graphDB,1), BorderLayout.CENTER);
                    revalidate();
                    repaint();
                }
        }

}
```

SubgraphPanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

import org.graphstream.graph.Graph;
import org.graphstream.stream.file.FileSinkImages;
import org.graphstream.stream.file.FileSinkImages.LayoutPolicy;
import org.graphstream.stream.file.FileSinkImages.OutputType;
import org.graphstream.stream.file.FileSinkImages.Resolutions;
import org.graphstream.ui.swingViewer.ViewPanel;
import org.graphstream.ui.view.Viewer;

import com.itextpdf.text.DocumentException;

import fpgraphminer.FPGfunctions;
import fpgraphminer.FPNode;
import model.Disease;
import pdfGenerator.ReportMaker;

public class SubgraphPanel extends JPanel implements MouseListener{

        private static final long serialVersionUID = 8112857964786902433L;
        private JPanel bg,options,output,top,bottom,genOutputpanel;
        private JLabel outputLabel;
        private JTextField userSupp;
        private JTextArea textDetail;
        private JButton back,export,show,fp;
        private JCheckBox willShowSignif;
        private Image panLogo,ZupLogo,ZdownLogo;
        private int width, height, counter;
        private double support;
        private ArrayList<Disease> graphDB;
        private FPGfunctions fpg;

        public SubgraphPanel(ArrayList<Disease> diseases, int width, int height){
                this.graphDB = diseases;
                this.width = width;
                this.height = height;
                this.counter = 0;
                this.fpg = new FPGfunctions(diseases);

                makePanel();
        }

        private void makePanel(){
                setLayout(new BorderLayout());

                setBG();
                setTop();
                setOptions();
                setOutput();
                setBottom();

                bg.add(top, BorderLayout.PAGE_START);
                bg.add(options,BorderLayout.LINE_START);
                bg.add(genOutputpanel, BorderLayout.CENTER);
                bg.add(bottom, BorderLayout.PAGE_END);
                add(bg, BorderLayout.CENTER);
                addMouseListener(this);
        }

        private void setTop(){
                top = new JPanel();
                top.setLayout(new BorderLayout());
                top.setOpaque(false);
                back = new JButton("Back");
                back.addMouseListener(this);
                top.add(back, BorderLayout.LINE_START);
        }

        private void setBG(){
                bg = new BackgroundPanel(getClass().getResource("/images/background.png"));
                bg.setLayout(new BorderLayout(10,10));
                bg.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        }

        private void setOptions(){
                options = new JPanel();
                options.setLayout(new BorderLayout(10,10));
                options.setOpaque(false);

                JLabel titleLabel = new JLabel("Input and Controls", SwingConstants.CENTER);
                titleLabel.setForeground(Color.WHITE);
                titleLabel.setFont(new Font("Arial",Font.PLAIN, 16));
```

```
TitledBorder paramBorder = BorderFactory.createTitledBorder(BorderFactory.
     createEtchedBorder(EtchedBorder.LOWERED), "Parameters");
paramBorder.setTitleFont(new Font("Arial",Font.PLAIN, 14));
paramBorder.setTitleColor(Color.WHITE);

JPanel parameters = new JPanel(new BorderLayout());
parameters.setPreferredSize(new Dimension(width/3,height/2));
parameters.setOpaque(false);

JPanel graphParam = new JPanel();
graphParam.setBackground(Color.WHITE);
graphParam.setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
graphParam.setPreferredSize(new Dimension(width/3,height/2));

gbc.anchor = GridBagConstraints.LINE_END;
gbc.gridx = 0;
gbc.gridy = 0;
graphParam.add(new JLabel("Support: "), gbc);
gbc.gridy++;
graphParam.add(new JLabel("Maximum Support: "), gbc);
gbc.gridy++;
graphParam.add(new JLabel("Default Support: "), gbc);

gbc.anchor = GridBagConstraints.PAGE_START;
gbc.gridx = 1;
gbc.gridy = 0;
userSupp = new JTextField(10);
graphParam.add(userSupp);
gbc.gridy++;
graphParam.add(new JLabel(Double.toString(fpg.getMaxSupport())), gbc);
gbc.gridy++;
graphParam.add(new JLabel("0.00"), gbc);
gbc.gridy++;
willShowSignif = new JCheckBox("Show component significance");
willShowSignif.setOpaque(false);
graphParam.add(willShowSignif,gbc);
gbc.gridy++;
show = new JButton("Show Subgraph");
show.addMouseListener(this);
graphParam.add(show, gbc);
parameters.add(graphParam, BorderLayout.CENTER);
parameters.setBorder(paramBorder);

JTabbedPane details = new JTabbedPane();
details.setPreferredSize(new Dimension(width/3, height/2));
details.setOpaque(false);
JPanel controls = new JPanel(new GridBagLayout());
controls.setBackground(Color.WHITE);
setControlImages();
JLabel panLabel = new JLabel(new ImageIcon(panLogo));
JLabel upLabel = new JLabel(new ImageIcon(ZupLogo));
JLabel downLabel = new JLabel(new ImageIcon(ZdownLogo));
controls.setPreferredSize(new Dimension(width/3, height/2));
TitledBorder contBorder = BorderFactory.createTitledBorder(BorderFactory.
     createEtchedBorder(EtchedBorder.LOWERED), "Details and Controls");
contBorder.setTitleFont(new Font("Arial",Font.PLAIN, 14));
contBorder.setTitleColor(Color.WHITE);
gbc.anchor = GridBagConstraints.LINE_START;
gbc.gridx = 0;
gbc.gridy = 0;
controls.add(new JLabel("Panning:"), gbc);
gbc.gridx = 1;
gbc.gridy++;
controls.add(panLabel, gbc);
gbc.gridx = 0;
gbc.gridy++;
controls.add(new JLabel("Zoom in: "), gbc);
gbc.gridx = 1;
gbc.gridy++;
controls.add(upLabel, gbc);
gbc.gridx = 0;
gbc.gridy++;
controls.add(new JLabel("Zoom out: "), gbc);
gbc.gridx = 1;
gbc.gridy++;
controls.add(downLabel, gbc);
contBorder.setTitleFont(new Font("Arial",Font.PLAIN, 14));
contBorder.setTitleJustification(TitledBorder.LEFT);

JPanel outputDetails = new JPanel(new BorderLayout());
textDetail = new JTextArea();
textDetail.setEditable(false);
textDetail.setFont(new Font("Arial",Font.PLAIN, 14));
textDetail.setText("Output details will be displayed here");
JScrollPane scrollable = new JScrollPane(textDetail,JScrollPane.
     VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
scrollable.setPreferredSize(new Dimension(width/3, height/2));
outputDetails.add(scrollable);

details.addTab("Details", outputDetails);
```

```
        details.addTab("Controls", controls);
        details.setBorder(contBorder);

        options.add(details, BorderLayout.PAGE_END);
        options.add(titleLabel, BorderLayout.PAGE_START);
        options.add(parameters, BorderLayout.CENTER);
}

private void setControlImages(){
        panLogo = null;ZupLogo = null;ZdownLogo = null;
        try {
                panLogo = ImageIO.read(getClass().getResource("/images/arrowkeys.png")).
                    getScaledInstance(width/10, height/6, Image.SCALE_SMOOTH);
                ZupLogo = ImageIO.read(getClass().getResource("/images/page-up.png")).
                    getScaledInstance(width/15, height/11, Image.SCALE_SMOOTH);
                ZdownLogo = ImageIO.read(getClass().getResource("/images/page-down.png")).
                    getScaledInstance(width/15, height/11, Image.SCALE_SMOOTH);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

private void setOutput(){
        genOutputpanel = new JPanel(new BorderLayout(10,10));
        genOutputpanel.setOpaque(false);

        outputLabel = new JLabel("Common Substructure", SwingConstants.CENTER);
        outputLabel.setForeground(Color.WHITE);
        outputLabel.setFont(new Font("Arial",Font.PLAIN, 14));

        output = new JPanel();
        output.setLayout(new BorderLayout());
        output.setPreferredSize(new Dimension(2*width/3,2*height/3));

        genOutputpanel.add(outputLabel, BorderLayout.PAGE_START);
        genOutputpanel.add(output, BorderLayout.CENTER);
}

private void setBottom(){
        bottom = new JPanel();
        export = new JButton("Export results as PDF");
        export.addMouseListener(this);
        fp = new JButton("View FP Table & Graph");
        fp.addMouseListener(this);
        bottom.add(export);
        bottom.add(fp);
        bottom.setOpaque(false);
}

@Override
public void mouseClicked(MouseEvent arg0) {}

@Override
public void mouseEntered(MouseEvent arg0) {}

@Override
public void mouseExited(MouseEvent arg0) {}

@Override
public void mousePressed(MouseEvent arg0) {}

@Override
public void mouseReleased(MouseEvent e) {
        if(e.getComponent().equals(back)){
                removeAll();
            add(new DiseaseToolPanel(graphDB,width,height), BorderLayout.CENTER);
            revalidate();
            repaint();
        }
        if(e.getComponent().equals(export)){
                String date = new SimpleDateFormat("MM-dd-yyyy HH,mm,ss").format(new Date
                    ());
                String filename = "[DSA] Substructure Search "+date+".pdf";
                ArrayList<String> imagePaths = new ArrayList<String>();
                ArrayList<Graph> graphs = fpg.getFrequentSubgraphsGraphStream(this.support
                    );
                ArrayList<ArrayList<String>> edgeList = fpg.getRelationTable();
                for(int graphCtr=0; graphCtr<graphs.size(); graphCtr++){
                        FileSinkImages pic = new FileSinkImages(OutputType.PNG,
                            Resolutions.VGA);
                        pic.setLayoutPolicy(LayoutPolicy.COMPUTED_FULLY_AT_NEW_IMAGE);
                        try {
                                pic.writeAll(graphs.get(graphCtr), "resources//exported
                                    images//Subgraph "+graphCtr+".png");
                                imagePaths.add("./resources//exported images//Subgraph "+
                                    graphCtr+".png");
                        } catch (IOException e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }
                }
                ReportMaker reportGen = new ReportMaker(filename, imagePaths, "Search
                    Frequent Subgraphs");
                try {
                        reportGen.setDiseaseNames(getDiseaseNames());
                        reportGen.makeSubgraphReport(edgeList, textDetail.getText());
                        JOptionPane.showMessageDialog(null, "Report generated", "Success
                            !", JOptionPane.INFORMATION_MESSAGE);
                } catch (FileNotFoundException | DocumentException e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                }

        }
        if(e.getComponent().equals(show)){
                try{
                        double support = Double.parseDouble(userSupp.getText());
                        this.support = support;
                        fpg.setSignif(willShowSignif.isSelected());
                        ArrayList<ArrayList<FPNode>> frequentSubgraphs = fpg.
                            getFrequentSubgraphs(support);
                        ArrayList<ArrayList<String>> queries = new ArrayList<ArrayList<
                            String>>();
                        for(ArrayList<FPNode> fpnodes : frequentSubgraphs){
                                ArrayList<String> query = new ArrayList<String>();
                                for(FPNode node: fpnodes){
                                        for(String[] tuple : node.getTuples()){
                                                query.add(tuple[0]);
                                        }
                                }
                                queries.add(query);
                        }
                        String content = "";
                        for(int qCtr=0; qCtr<queries.size(); qCtr++){
                                content = content+"Subgraph "+(qCtr+1)+" is in the
                                    following diseases:\n";
                                ArrayList<String> q = queries.get(qCtr);
                                ArrayList<Disease> diseases = fpg.getGraphsContainingQuery
                                    (q);
                                for(Disease disease: diseases){
                                        content = content+"\t>"+disease.getTitle()+"\n";
                                }
                        }
                        textDetail.setText(content);
                        int subgraphCount = fpg.getSubgraphCount();
                        output.removeAll();
                        Graph subgraph;
                        if(counter>=subgraphCount){
                                counter = 0;
                                subgraph = fpg.showFrequentSubgraphs(frequentSubgraphs,
                                    counter);
                                if(subgraphCount!=1){
                                        show.setText("Show next subgraph");
                                        outputLabel.setText("Common Substructure "+(
                                            counter+1));
                                }
                                counter++;
                        }else{
                                subgraph = fpg.showFrequentSubgraphs(frequentSubgraphs,
                                    counter);
                                if(subgraphCount!=1){
                                        show.setText("Show next subgraph");
                                        outputLabel.setText("Common Substructure "+(
                                            counter+1));
                                }
                                counter++;
                        }
                        Viewer viewer = new Viewer(subgraph, Viewer.ThreadingModel.
                            GRAPH_IN_ANOTHER_THREAD);
                    ViewPanel viewPanel = viewer.addDefaultView(false);
                    viewPanel.setPreferredSize(output.getPreferredSize());
                    viewer.enableAutoLayout();
                    output.add(viewPanel, BorderLayout.CENTER);
                        revalidate();
                    repaint();
                }catch(NullPointerException n){
                        JOptionPane.showMessageDialog(null, "No subgraphs found. Support
                            exceeds the maximum possible value.", "Error", JOptionPane.
                            ERROR_MESSAGE);
                }catch(NumberFormatException n){
                        JOptionPane.showMessageDialog(null, "Please specify a support
                            value", "Error", JOptionPane.ERROR_MESSAGE);
                }
        }if(e.getComponent().equals(fp)){
                removeAll();
            add(new FPpanel(width,height,this.graphDB,2), BorderLayout.CENTER);
            revalidate();
            repaint();
        }
}


private ArrayList<String> getDiseaseNames(){
```

```
                ArrayList<String> names = new ArrayList<String>();
                for(Disease d:this.graphDB){
                        names.add(d.getTitle());
                }
                return names;
        }
}
```

FPPanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Random;
import java.util.Set;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.SwingConstants;

import org.graphstream.graph.Graph;
import org.graphstream.graph.Node;
import org.graphstream.graph.implementations.SingleGraph;
import org.graphstream.ui.swingViewer.ViewPanel;
import org.graphstream.ui.view.Viewer;

import fpgraphminer.FPCluster;
import fpgraphminer.FPGfunctions;
import fpgraphminer.FPGraph;
import fpgraphminer.FPGraphMiner;
import fpgraphminer.FPNode;
import model.Disease;

public class FPpanel extends JPanel{

        private static final long serialVersionUID = 32515624471871485L;

        public FPpanel(int width, int height, ArrayList<Disease> diseases, int functionality){
                setLayout(new BorderLayout());

                FPGraph fpg = new FPGraph(diseases);
                FPGfunctions fpgfunc = new FPGfunctions(diseases);
                ArrayList<FPCluster> clusters = fpg.createFPGraph(fpg.getDistinctEdges(diseases),
                    fpg.getDistinctDisconnectedNodes(diseases));
                ArrayList<String[]> fptable = fpg.getFPTable();
                Hashtable<String,String> relEncoding = fpg.getEncoding();
                Hashtable<String,String> compEncoding = fpgfunc.distinctCompEncoding();

                JPanel bg = new BackgroundPanel(getClass().getResource("/images/background.png"));
                bg.setLayout(new BorderLayout(10,10));
                bg.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

                JPanel signifPanel = new JPanel(new BorderLayout());
                JTable signifTable = makeSignifTable(clusters, compEncoding);
                JLabel signifLabel = new JLabel("Significance Table", SwingConstants.CENTER);
                JScrollPane signifScroll =  new JScrollPane(signifTable);
                signifLabel.setForeground(Color.WHITE);
                signifLabel.setFont(new Font("Arial",Font.PLAIN, 14));
                signifPanel.setPreferredSize(new Dimension(width/4, height));
                signifPanel.setOpaque(false);
                signifScroll.setPreferredSize(new Dimension(width/4, height));
                signifPanel.add(signifLabel, BorderLayout.PAGE_START);
                signifPanel.add(signifScroll, BorderLayout.CENTER);

                JPanel tablePanel = new JPanel(new BorderLayout());
                JTable table = makeFPTable(fptable,relEncoding);
                JLabel tableLabel = new JLabel("FP Table", SwingConstants.CENTER);
                tableLabel.setForeground(Color.WHITE);
                tableLabel.setFont(new Font("Arial",Font.PLAIN, 14));
                JScrollPane tableScroll = new JScrollPane(table);
                tableScroll.setPreferredSize(new Dimension(width/4, height));
                tablePanel.setPreferredSize(new Dimension(width/4, height));
                tablePanel.setOpaque(false);
                tablePanel.add(tableLabel, BorderLayout.PAGE_START);
                tablePanel.add(tableScroll, BorderLayout.CENTER);


                JPanel graphPanel = new JPanel(new BorderLayout(10,10));
                JLabel graphLabel = new JLabel("FP Graph", SwingConstants.CENTER);
```

```
            JPanel graphCont = new JPanel(new BorderLayout());
            graphLabel.setForeground(Color.WHITE);
            graphLabel.setFont(new Font("Arial",Font.PLAIN, 14));
            graphPanel.setPreferredSize(new Dimension(width/2, height));
            Graph graph = makeFPGraph(clusters);
            Viewer viewer = new Viewer(graph, Viewer.ThreadingModel.GRAPH_IN_ANOTHER_THREAD);
        ViewPanel viewPanel = viewer.addDefaultView(false);
        viewPanel.setPreferredSize(graphPanel.getPreferredSize());
        viewer.enableAutoLayout();
        graphCont.add(viewPanel, BorderLayout.CENTER);
        graphPanel.setOpaque(false);
        graphPanel.add(graphLabel, BorderLayout.PAGE_START);
        graphPanel.add(graphCont, BorderLayout.CENTER);
            revalidate();
        repaint();


            JPanel bottom = new JPanel();
            JButton back = new JButton("Back");
            back.addMouseListener(new MouseAdapter(){
                    public void mousePressed(MouseEvent mouse){
                            removeAll();
                            if(functionality == 1){
                                    add(new GraphSearchPanel(diseases,width,height),
                                         BorderLayout.CENTER);
                            }else if(functionality == 2){
                                    add(new SubgraphPanel(diseases,width,height), BorderLayout
                                         .CENTER);
                            }
                        revalidate();
                        repaint();
                    }
            });
            bottom.setOpaque(false);
            bottom.add(back);

            bg.add(tablePanel, BorderLayout.LINE_START);
            bg.add(graphPanel, BorderLayout.CENTER);
            bg.add(signifPanel, BorderLayout.LINE_END);
            bg.add(bottom, BorderLayout.PAGE_END);
            add(bg, BorderLayout.CENTER);
    }

    private JTable makeSignifTable(ArrayList<FPCluster> fpgraph, Hashtable<String,String>
         encoding) {
            String[] headers = {"Component","Weighted Average"};
            FPGraphMiner miner = new FPGraphMiner();
            Hashtable<String,Double> signif = miner.getWeightedAverages(fpgraph);
            String[][] data = new String[signif.size()][2];
            Set<String> keySet = signif.keySet();
            int rowCtr = 0;
            for(String key : keySet){
                    data[rowCtr][0] = encoding.get(key);
                    data[rowCtr][1] = Double.toString(signif.get(key));
                    rowCtr++;
            }
            JTable table = new JTable(data,headers){
                    private static final long serialVersionUID = 320073743100325790L;
                    public boolean isCellEditable(int row, int column) {
            return false;
                    };
            };
            return table;
    }

    private JTable makeFPTable(ArrayList<String[]> fpt,Hashtable<String,String> encoding){
            String[][] fptArr = new String[fpt.size()][2];
            for(int ctr=0; ctr<fpt.size(); ctr++){
                    fptArr[ctr][0] = encoding.get(fpt.get(ctr)[0]);
                    fptArr[ctr][1] = fpt.get(ctr)[1];
            }
            String[] labels = {"Edge names","BitCode"};
            JTable table = new JTable(fptArr,labels){
                    private static final long serialVersionUID = 320073743100325790L;
                    public boolean isCellEditable(int row, int column) {
            return false;
                    };
            };
            return table;
    }

    private Graph makeFPGraph(ArrayList<FPCluster> clusters){
            Graph graph = new SingleGraph("FP Graph");
            ArrayList<FPNode> nodes = new ArrayList<FPNode>();
            ArrayList<String> edgeNames = new ArrayList<String>();
            for(FPCluster cluster : clusters){
                    Random rand = new Random();
                int r = rand.nextInt(156) + 100;
                int g = rand.nextInt(156) + 100;
                int b = rand.nextInt(156) + 100;
                    for(FPNode node : cluster.getNodes()){
                            nodes.add(node);
```

```
                                        graph.addNode(node.getBitcode());
                                }
                                for(int nCtr=0;nCtr<cluster.getNodes().size()-1; nCtr++){
                                        for(int ctr=nCtr+1;ctr<cluster.getNodes().size(); ctr++){
                                                FPNode node1 = cluster.getNodes().get(nCtr);
                                                FPNode node2 = cluster.getNodes().get(ctr);
                                                Node gNode1 = graph.getNode(node1.getBitcode());
                                                graph.addEdge(node1.getBitcode()+"-"+node2.getBitcode(),
                                                     gNode1, graph.getNode(node2.getBitcode()));
                                                graph.getEdge(node1.getBitcode()+"-"+node2.getBitcode()).
                                                     addAttribute("ui.style", "fill-color: rgb("+r+","+g
                                                     +","+b+"); shape: blob;");
                                        }
                                }
                        }
                        graph.addNode("Header node");
                        for(Node node:graph){
                                node.setAttribute("ui.label", node.getId());
                                node.setAttribute("ui.style", "text-alignment: at-left;");
                        }
                        for(FPNode node : nodes){
                                Node gNode = graph.getNode(node.getBitcode());
                                for(FPNode connNode : node.getConnectedNodes()){
                                        if(connNode.getBitcode() != null){//header
                                                if(!edgeNames.contains(node.getBitcode()+"-"+connNode.
                                                     getBitcode())
                                                                && !edgeNames.contains(connNode.getBitcode
                                                                     ()+"-"+node.getBitcode())){
                                                        edgeNames.add(node.getBitcode()+"-"+connNode.
                                                             getBitcode());
                                                        edgeNames.add(connNode.getBitcode()+"-"+node.
                                                             getBitcode());
                                                        graph.addEdge(node.getBitcode()+"-"+connNode.
                                                             getBitcode(), gNode, graph.getNode(connNode.
                                                             getBitcode()));
                                                }
                                        }
                                }
                                if(node.isConnectedtoHeader()){
                                        edgeNames.add(node.getBitcode()+"-header");
                                        edgeNames.add("header-"+node.getBitcode());
                                        graph.addEdge("header-"+node.getBitcode(), gNode, graph.getNode("
                                             Header node"));

                                }
                        }
                        graph.addAttribute("ui.antialias");
                        return graph;
                }

}
```

DBPanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.File;
import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;

import javax.swing.BorderFactory;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ListSelectionModel;
import javax.swing.SwingConstants;
import javax.swing.SwingWorker;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;
import javax.swing.filechooser.FileSystemView;

import model.Database;
import model.Disease;
import model.KGMLReader;
import restKEGG.RestKEGG;

public class DBPanel extends JPanel implements MouseListener{

        private static final long serialVersionUID = 8356035331980871569L;
        private JPanel bg,currDiseases;
        private JButton addfromPC, addfromNet, delete, back, addSelected;
        private JScrollPane scrollAvailableDiseases;
        private JProgressBar progress;
        private JList<String> availableDiseases,keggPathwayList;
        private int width, height;
        private Database db;
        private KGMLReader filereader;
        private DefaultListModel<String> listModel;
        private ArrayList<String[]> pathways;
        private ArrayList<String> diseaseKeggCode;

        public DBPanel(int w, int h){
                this.width = w;
                this.height = h;
                this.db = new Database();
                this.filereader = new KGMLReader();
                setLayout(new BorderLayout());

                bg = new BackgroundPanel(getClass().getResource("/images/background.png"));
                bg.setLayout(new BorderLayout(10,10));
                bg.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

                setNotePanel();
                setCurrDiseasesPanel();

                bg.add(currDiseases, BorderLayout.CENTER);
                add(bg, BorderLayout.CENTER);
                addMouseListener(this);
        }

        private void setCurrDiseasesPanel(){
                currDiseases = new JPanel(new BorderLayout(10,10));
                currDiseases.setPreferredSize(new Dimension((4*width)/5,height));
                currDiseases.setOpaque(false);

                db = new Database();
                String[][] diseaseTuples = db.getDiseaseNamesWithKEGGCode();
                String[] diseaseList = getCommonNames(diseaseTuples);
                diseaseKeggCode = getCodeNames(diseaseTuples);
                listModel = new DefaultListModel<String>();
                availableDiseases = new JList<String>();
                for(String disease : diseaseList){
                        listModel.addElement(disease);
                }
                availableDiseases.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION)
                    ;
                availableDiseases.setFont(new Font("Arial",Font.PLAIN, 14));
                availableDiseases.setModel(listModel);
                scrollAvailableDiseases = new JScrollPane(availableDiseases);
                scrollAvailableDiseases.setPreferredSize(new Dimension(4*currDiseases.
                    getPreferredSize().width/5,8*currDiseases.getPreferredSize().height/10));

                JPanel titlePanel = new JPanel(new BorderLayout());
                back = new JButton("Back");
                back.addMouseListener(this);
                titlePanel.add(back, BorderLayout.LINE_START);
                titlePanel.setOpaque(false);

                JLabel title = new JLabel("Currently Stored Disease Pathways", SwingConstants.
                    CENTER);
                title.setFont(new Font("Arial",Font.PLAIN, 14));
                title.setForeground(Color.WHITE);

                JPanel localDB = new JPanel(new BorderLayout());
                localDB.setOpaque(false);
                localDB.add(scrollAvailableDiseases,BorderLayout.CENTER);
                localDB.add(title,BorderLayout.PAGE_START);
                currDiseases.add(localDB, BorderLayout.CENTER);
                bg.add(titlePanel, BorderLayout.PAGE_START);
        }

        private void setNotePanel(){
                JPanel notes = new JPanel(new BorderLayout(10,10));
                notes.setPreferredSize(new Dimension(width/5,height));
                notes.setOpaque(false);
                bg.add(notes, BorderLayout.LINE_START);

                JPanel buttonPanel = new JPanel();
                buttonPanel.setLayout(new GridLayout(4,1));
                buttonPanel.setPreferredSize(new Dimension(width/5, height/5));
                TitledBorder buttonBorder = BorderFactory.createTitledBorder(BorderFactory.
                    createEtchedBorder(EtchedBorder.LOWERED), "Actions");
                buttonBorder.setTitleFont(new Font("Arial",Font.PLAIN, 14));
                buttonBorder.setTitleColor(Color.WHITE);
                buttonPanel.setBorder(buttonBorder);
                buttonPanel.setOpaque(false);
                Dimension buttonDim = new Dimension(width/4,height/20);
                addfromPC = new JButton("Add kgml from PC");
                addfromNet = new JButton("Show available kgml from KEGG");
                delete = new JButton("Delete disease");
                addSelected = new JButton("Add Selected Diseases");
                addSelected.setEnabled(false);
                addfromPC.setPreferredSize(buttonDim);
                addfromNet.setPreferredSize(buttonDim);
                addSelected.setPreferredSize(buttonDim);
                delete.setPreferredSize(buttonDim);
                addfromPC.addMouseListener(this);
                addfromNet.addMouseListener(this);
                delete.addMouseListener(this);
                addSelected.addMouseListener(this);
                buttonPanel.add(addfromPC);
                buttonPanel.add(addfromNet);
                buttonPanel.add(addSelected);
                buttonPanel.add(delete);

                JPanel reminders = new JPanel(new BorderLayout());
                reminders.setOpaque(false);
                TitledBorder remBorder = BorderFactory.createTitledBorder(BorderFactory.
                     createEtchedBorder(EtchedBorder.LOWERED), "Notes");
                remBorder.setTitleFont(new Font("Arial",Font.PLAIN, 14));
                remBorder.setTitleColor(Color.WHITE);
                JTextArea text = new JTextArea();
                text.setEditable(false);
                text.setLineWrap(true);
                text.setWrapStyleWord(true);
                text.setText("Please do NOT add tampered kgml files.\n\nWhen adding diseases from
                     Kyoto Encyclopedia of Genes and Genomes(KEGG), please note that NOT all of the
                      pathways listed are disease pathways.");
                text.setFont(new Font("Arial",Font.PLAIN, 13));
                reminders.add(text);
                reminders.setBorder(remBorder);

                progress = new JProgressBar(JProgressBar.HORIZONTAL);
                progress.setVisible(false);

                notes.add(buttonPanel, BorderLayout.PAGE_START);
                notes.add(reminders, BorderLayout.CENTER);
                notes.add(progress, BorderLayout.PAGE_END);
        }

        @Override
        public void mouseClicked(MouseEvent arg0) {}

        @Override
        public void mouseEntered(MouseEvent arg0) {}

        @Override
        public void mouseExited(MouseEvent arg0) {}

        @Override
        public void mousePressed(MouseEvent arg0) {}

        @Override
        public void mouseReleased(MouseEvent mouse) {
                if(mouse.getComponent().equals(addfromPC)){
                        JFileChooser fileChooser = new JFileChooser(FileSystemView.
                             getFileSystemView().getHomeDirectory());
                        fileChooser.setMultiSelectionEnabled(true);
                        fileChooser.showOpenDialog(null);
                        File[] kgmlFiles = fileChooser.getSelectedFiles();
                        ArrayList<String> savedDiseases = new ArrayList<String>();

                        for(int elementCtr=0; elementCtr<listModel.size(); elementCtr++){
                                savedDiseases.add(listModel.getElementAt(elementCtr));
                        }

                        class Worker extends SwingWorker<String, Object>{
                                protected String doInBackground() {
                                        progress.setVisible(true);
                                        progress.setIndeterminate(true);
                                        addfromPC.setEnabled(false);
                                        addfromNet.setEnabled(false);
                                        delete.setEnabled(false);
                                        back.setEnabled(false);

                               for(File kgmlFile : kgmlFiles){
                                                String absPath = kgmlFile.getAbsolutePath();
                                                Disease disease = filereader.readXML(absPath);
                                                db.insertDisease(disease);
                                                if(!savedDiseases.contains(disease.getTitle())){
                                                        listModel.addElement(disease.getTitle());
                                                }
                                                if(!diseaseKeggCode.contains(disease.getName())){
                                                        diseaseKeggCode.add(disease.getName());
                                                }
                                        }
                               return "Done.";
                             }
                        protected void done() {
                            progress.setVisible(false);
                            addfromPC.setEnabled(true);
                            addfromNet.setEnabled(true);
                            delete.setEnabled(true);
                            back.setEnabled(true);
                        }
                    }
                    Worker worker = new Worker();
                    worker.execute();
            }else if(mouse.getComponent().equals(addfromNet)){
                    RestKEGG rest = new RestKEGG();
                    pathways = new ArrayList<String[]>();
                    class Worker extends SwingWorker<String, Object>{
                        protected String doInBackground() {
                            progress.setVisible(true);
                          progress.setIndeterminate(true);
                          addfromPC.setEnabled(false);
                          addfromNet.setEnabled(false);
                          delete.setEnabled(false);
                          back.setEnabled(false);

                          try {
                              pathways = rest.getPathwaysfromKEGG();
                          } catch (IOException e) {
                              JOptionPane.showMessageDialog(null, "IO Error. Please check
                                   your internet connection", "IO Error", JOptionPane.
                                   ERROR_MESSAGE);
                          }
                          return "Done.";
                        }

                        protected void done() {
                           progress.setVisible(false);
                           addfromPC.setEnabled(true);
                           addfromNet.setEnabled(true);
                           delete.setEnabled(true);
                           back.setEnabled(true);
                           addSelected.setEnabled(true);
                           JPanel keggPathwayPanel = new JPanel(new BorderLayout());
                                   keggPathwayPanel.setOpaque(false);
                                   keggPathwayPanel.setPreferredSize(new Dimension(2*width/5,
                                        height));
                                   currDiseases.setPreferredSize(new Dimension(2*width/5,
                                        height));

                                   JLabel title = new JLabel("Pathways available in KEGG",
                                        SwingConstants.CENTER);
                                   title.setFont(new Font("Arial",Font.PLAIN, 14));
                                   title.setForeground(Color.WHITE);
                                   keggPathwayList = new JList<String>(getNames(pathways));
                                   keggPathwayList.setFont(new Font("Arial",Font.PLAIN, 14));
                                   JScrollPane scrollable = new JScrollPane(keggPathwayList);

                                   keggPathwayPanel.add(title, BorderLayout.PAGE_START);
                                   keggPathwayPanel.add(scrollable, BorderLayout.CENTER);
                                   bg.add(keggPathwayPanel, BorderLayout.LINE_END);
                        }
                    }

                    Worker worker = new Worker();
                    worker.execute();
                    revalidate();
                    repaint();
            }else if(mouse.getComponent().equals(delete)){
                    try {
                           ArrayList<String> deleteList = (ArrayList<String>)
                                availableDiseases.getSelectedValuesList();
                           int[] deleteIndex = availableDiseases.getSelectedIndices();
                           db.deleteDisease(deleteList);
                           for(int delCtr=0; delCtr<deleteList.size(); delCtr++){
                                   listModel.removeElement(deleteList.get(delCtr));
                           }
                           for(int index : deleteIndex){
                                   File deleteImg = new File("/resources//pathway images//"+
                                        diseaseKeggCode.get(index)+".png");
                                   deleteImg.delete();
                           }
                           JOptionPane.showMessageDialog(null, "Finished deleting the disease
                                 data", "Done!", JOptionPane.INFORMATION_MESSAGE);
                    } catch (SQLException e) {
                           JOptionPane.showMessageDialog(null, "Deletion Failed!", "Failed",
                                JOptionPane.ERROR_MESSAGE);
                    } catch (ClassCastException clEx){
                           JOptionPane.showMessageDialog(null, "Empty selection", "Error",
                                JOptionPane.ERROR_MESSAGE);
                    }
            }else if(mouse.getComponent().equals(addSelected)){
                    File imageFolder = new File("resources//pathway images");
                    boolean imgDirexists = imageFolder.exists();
                    ArrayList<String> savedDiseases = new ArrayList<String>();
```

```
                    for(int elementCtr=0; elementCtr<listModel.size(); elementCtr++){
                            savedDiseases.add(listModel.getElementAt(elementCtr));
                    }
                    class Worker extends SwingWorker<String, Object>{
                         protected String doInBackground() {
                           progress.setVisible(true);
                           progress.setIndeterminate(true);
                           addfromPC.setEnabled(false);
                           addfromNet.setEnabled(false);
                           delete.setEnabled(false);
                           back.setEnabled(false);
                           addSelected.setEnabled(false);

                           RestKEGG rest = new RestKEGG();
                                    int[] indices = keggPathwayList.getSelectedIndices();
                                    for(int ctr=0; ctr<indices.length; ctr++){
                                            Disease disease = filereader.makeDisease(rest.
                                                 readXMLfromKEGGURL("http://rest.kegg.jp/get/"+
                                                 pathways.get(indices[ctr])[0]+"/kgml"));
                                            db.insertDisease(disease);
                                            if(!savedDiseases.contains(disease.getTitle())){
                                                    listModel.addElement(disease.getTitle());
                                            }
                                            if(!diseaseKeggCode.contains(disease.getName())){
                                                    diseaseKeggCode.add(disease.getName());
                                            }
                                            if(imgDirexists){
                                                    rest.downloadImg("http://rest.kegg.jp/get
                                                         /"+pathways.get(indices[ctr])[0]+"/
                                                         image", pathways.get(indices[ctr])[0])
                                                         ;
                                            }else{
                                                    boolean makeImageDir = new File("resources
                                                         //pathway images").mkdirs();
                                                    if(makeImageDir){
                                                            rest.downloadImg("http://rest.kegg
                                                                 .jp/get/"+pathways.get(indices
                                                                 [ctr])[0]+"/image", pathways.
                                                                 get(indices[ctr])[0]);
                                                    }
                                            }
                                    }

                           return "Done.";
                         }

                         protected void done() {
                            progress.setVisible(false);
                            addfromPC.setEnabled(true);
                            addfromNet.setEnabled(true);
                            delete.setEnabled(true);
                            back.setEnabled(true);
                            addSelected.setEnabled(true);
                            JOptionPane.showMessageDialog(null, "Finished adding the selected
                                 diseases", "Success", JOptionPane.INFORMATION_MESSAGE);
                         }
                      }

                    Worker worker = new Worker();
                    worker.execute();
            }
            else if(mouse.getComponent().equals(back)){
                    removeAll();
                add(new MainPanel(width,height), BorderLayout.CENTER);
                revalidate();
                repaint();
            }
    }

    private String[] getNames(ArrayList<String[]> pathways){
            String[] names = new String[pathways.size()];
            for(int ctr=0; ctr<names.length; ctr++){
                    names[ctr] = pathways.get(ctr)[1];
            }
            return names;
    }

    private ArrayList<String> getCodeNames(String[][] fromDB){
            ArrayList<String> codeNames = new ArrayList<String>();
            for(int ctr=0; ctr<fromDB.length; ctr++){
                    codeNames.add(fromDB[ctr][1]);
            }
            return codeNames;
    }

    private String[] getCommonNames(String[][] fromDB){
            String[] commonNames = new String[fromDB.length];
            for(int ctr=0; ctr<fromDB.length; ctr++){
                    commonNames[ctr] = fromDB[ctr][0];
            }
            return commonNames;
    }
```

```
}
```

UserGuidePanel.java

```
package view;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;

public class UserGuidePanel extends JPanel{

        /**
         *
         */
        private static final long serialVersionUID = 3344202584241051696L;
        private JPanel bg;

        public UserGuidePanel(int w, int h){
                setLayout(new BorderLayout());
                bg = new BackgroundPanel(getClass().getResource("/images/background.png"));
                bg.setPreferredSize(new Dimension(w,h));
                bg.setLayout(new BorderLayout(5,5));
                bg.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
                JTabbedPane tabbedPane = new JTabbedPane();
                tabbedPane.setTabPlacement(JTabbedPane.LEFT);
                JPanel bottom = new JPanel();
                bottom.setLayout(new FlowLayout());
                JPanel panel1 = new JPanel(new BorderLayout());
                tabbedPane.addTab("Clustering", panel1);
                JTextArea text1 = new JTextArea(2*w/3,h);
                text1.setEditable(false);
                text1.setLineWrap(true);
                text1.setWrapStyleWord(true);
                text1.setText("Linkage: \n\t The available linkages that can used by this tool are
                      Single Linkage, Average Linkage and Complete Linkage. Single, average and
                     complete linkage computes the distance between two elements by getting their
                     minimum, average and maximum distance respectively."+
                                        "\n\nMetric: \n\t There are two similarity metrics that
                                             can be used to quantify the elements' similarity
                                             namely Jaccard Similarity Index and Hamming Distance.
                                             Jaccard Similarity Index(JSI) defines the similarity
                                             between two entities as the quotient between
                                             intersection and the union of the entities' specified
                                             attribute, which in this case is the gene and compound
                                              relations of the disease pathways. Hamming Distance
                                             defines the similarity as the number of different
                                             elements between the two entities."
                                        +"\n\n*For set wise comparison, use JSI and for element(
                                             biological relation) wise comparison, use hamming
                                             distance.\n\n"+
                                        "Exporting the clustering output:\n\t When exporting the
                                             output, make sure that the needed details are
                                             currently displayed in the output panel by panning and
                                              zooming.");
                panel1.add(text1, BorderLayout.CENTER);

                JPanel panel2 = new JPanel(new BorderLayout());
                tabbedPane.addTab("Subgraph Search", panel2);
                JTextArea text2 = new JTextArea(2*w/3,h);
                text2.setEditable(false);
                text2.setLineWrap(true);
                text2.setWrapStyleWord(true);
                text2.setText("Support:\n\t Support is defined by the percentage of the graphs(
                     diseases) that contains the displayed subgraph or substructure.\n\n\t"
                                +"Aside from the controls that can be seen on the left side of the
                                      subgraph search panel, the user can also move around the
                                     nodes of the displayed graph to view the components by
                                     dragging them.");
                panel2.add(text2, BorderLayout.CENTER);

                JPanel panel3 = new JPanel(new BorderLayout());
                tabbedPane.addTab("Disease Search", panel3);
                JTextArea text3 = new JTextArea(2*w/3,h);
                text3.setEditable(false);
                text3.setLineWrap(true);
                text3.setWrapStyleWord(true);
                text3.setText("This functionality returns the disease/s that contains the user
                     selected relation/s.\n\nThe user query will be highlighted in red in the
                     displayed graph and aside from the controls that can be seen on the left side
                     of the subgraph search panel, the user can also move around the nodes of the
                     displayed graph to view the components by dragging them.");
                panel3.add(text3, BorderLayout.CENTER);

                JButton proceed = new JButton("Back to main screen");
                proceed.addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent mousePress){
                                removeAll();
                            add(new MainPanel(w,h), BorderLayout.CENTER);
                            revalidate();
                            repaint();
                        }
                });

                JButton menu = new JButton("Back to Start screen");
                menu.addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent mousePress){
                                removeAll();
                            add(new DiseaseSelectionPanel(w,h), BorderLayout.CENTER);
                            revalidate();
                            repaint();
                        }
                });
                bottom.add(proceed);
                bottom.setOpaque(false);
                bg.add(bottom, BorderLayout.AFTER_LAST_LINE);
                bg.add(tabbedPane, BorderLayout.CENTER);
                add(bg);
        }

}
```

XI. Acknowledgement

This is it, the final chapter of this one hell of a problem. Let me just thank everyone who helped me throughout the whole process of making this requirement.

First of all, thank you God for answering my prayers. Even if I wasn't exactly a good child, You still blessed me with everything that I need to finish this project. I would also like to thank my family, from my parents to my aunt and her family, for all the support that they gave me.

To my special problem adviser Sir. Geoffrey Solano, thank you for entrusting me this topic. Thank you for guiding me until I finished this requirement and I'm sorry if I disappointed you somewhere along the way. The tips that you gave us helped in keeping us in the right track and I'm sure that I can use the same tips whenever I'm facing another challenging problem, so again, thank you sir. To Dr. Angelyn Lao, thank you for the few yet fruitful meetings that we had. All of your comments and suggestions regarding this topic greatly helped me not just in finishing but also improving the output that I made. To my other professors, thank you for imparting your knowledge to us. Special mention (and thanks) to my math and computer science professors, Ma'am Basco, Ma'am Hermie, Sir Mong, Sir Ignacio, Sir Althom, etc. who made me like the field that I'm pursuing.

To all my co-advisees, thank you for helping me in finishing this requirement. All of our discussions gave me new ideas or approaches that helped me solve the problems I faced while working on this one. Ok, here goes the individual thanks. To Bianca Silmaro, thank you for opening the door to becoming Sir Solano's advisee. Without that offer, none of this would've happened. All of the discussions we had about your project taught me a lot and helped me in improving my own software. To Kharl Agir, thanks for being around even when I'm being annoying as hell and for accommodating me for several days in your home. Thank you for trusting and relying on me when you needed help as those moments also became a learning experience to me. To Cess Florendo, thank you also for accommodating us in your home several times, for answering all of my random questions for the past few semesters and for the unlimited moral support. To Bengemin Uy (not my co-advisee), thank you for helping me throughout my whole college life, especially during the first few semesters where I literally didn't know what the hell am I doing. Thank you also for listening and giving your opinions to me when I talk about my SP as your inputs always give me new ways to view the problem that I'm solving. To kuya James Sarmiento, thank you for answering my questions and when I was beginning to work on my SP.

To that one high school friend, thank you for always reminding me to keep my health in check and for the moral support that you gave me. I will not forget how happy you were when I told you that I'm almost done with my college life. So yeah, thank you.

To Aimer, One Ok Rock, supercell, EGOIST, Daughtry and SPYAIR, thank you for your amazing music. Your works helped relax, focus while working on my SP, kept me up and accompanied me through all of those sleepless nights. You don't know how much your songs helped me. Even though all of you can't see this, I would still like to thank everyone. To MagikarpUsedFly, Bricky and TouhouSniper98, thank you for all the quality content and memes that helped me keep my sanity.

I simply cannot mention everyone here so, for everyone who helped me that I didn't or I forgot to acknowledge, thank you. I appreciate all the help.

That's it. Peace out.