CoVi - a JAVA application to explore Human Disease Networks

Klaus Lyko
lyko.klaus@gmail.com

Victor Christen
vicolinho@gmail.com

Anastasia Chyhir
guapolife@gmail.com

University of Leipzig, Bachelor’s programme in Computer Science

Abstract: This paper presents the Comorbidity Viewer (CoVi), a platform independent Java application used to visualize Human Disease Networks. These networks encompass correlations between human diseases. CoVi helps to explore these publicly accessible datasets using an intuitive graphical user interface. Furthermore, it is capable of clustering disease networks by using the Borderflow algorithm.

1 Introduction

Human diseases rarely occur isolated from one another. Individuals are often affected concurrently by different diseases. Such a phenomenon is called comorbidity. Comorbid affections can have several causes. A genotypic relationship between diseases exists if the diseases are associated with the same gene. In addition, relationships of diseases via protein interactions are known. It is a common procedure to study those correlations with a network approach and model them as Human Disease Networks [GCV+07]. Hidalgo et al. analyzed over 30 million medical records of American hospitals and studied the prevalence of concurrent diseases as a Phenotypic Disease Network [HBBC09]. With this data it is possible to study comorbidity from a phenotypic point of view. Researchers, doctors, or patients can discover interactions of diseases, study their development, and analyze the data in combination with other variables, such as sex and race. Therefore, this data is of great interest to medical experts.

Hidalgo et al. analyzed the comorbidity of diseases solely based on contemporaneous diagnoses of diseases [HBBC09]. They developed a flash application called HuDiNe providing a representation of the collected data as a Human Disease Network. The goal of our project was to develop a platform-independent desktop application to display and explore these HuDiNe datasets with largely the same features as in their visualization. Here we present CoVi (Comorbidity Viewer), enabling users to search for a certain dis-

1http://hudine.neu.edu/resource/data/data.html
2http://barabasilab.neu.edu/projects/hudine/
3http://pcai042.informatik.uni-leipzig.de/~swp10-8/
ease in loaded HuDiNe datasets, to discover other diseases connected and further to explore the network in an intuitive, graphical way. Furthermore, CoVi uses the clustering algorithm Borderflow [NS09], which makes it possible to cluster a given disease network.

The paper is structured as follows: After discussing related work in Section 2 we describe the data and our approach to visualize it. Section 4 presents the features, architecture, and workflow of CoVi. In Section 5 we briefly discuss the performance of our program.

2 Related Work

The comorbidity between diseases can be understood as a network of those diseases connected by their comorbid relationship. Such a representation as a network provides users a more intuitive access to this data [Noe09]. Thus, it allows them to discover connections between diseases more easily. There are many information visualization tools for a broad range of applications [HMM00, Noe09].

The SONIVIS project\(^4\) allows the representation of different networks and provides a huge range of opportunities, such as support of different network definitions, filter functions for specific analysis of given categories, and clustering of attribute data. Even though this project is not part of the medical domain, it illustrates the opportunities of network modeling, which can also be used for disease visualization. The open source graph visualization tool Gephi [BHJ09] supports a wide variety of networks from different domains and could be customized to support the HuDiNe format as source data as well. It allows the user to manipulate the visualization in many ways and has built-in clustering mechanisms. As it uses the 3D render engine to visualize networks in real-time, it supports a fast exploration experience even in very large graphs.

In the medical domain many visualization tools exist for different purposes. The Diseasome project\(^5\) is most notable for phenotypic as well as genotypic Human Disease Networks [GCV07]. It not only holds information about disease-disease interactions, but also provides access to connections between diseases via genes. Gene connections complicate the graph visualization for the ordinary user and have influence on the time required to display the network. Additionally, due to its specific domain the Diseasome project has no wide layout functionality and it does not support clustering. This is a typical issue for visualization, because “[…] there is no such thing as a general-purpose visualization style for all networks, not even in the same area […] Good network visualizations are tailored for a specific task and a specific network at hand.” [Noe09, p.16].

\(^4\)http:sonivis.org/
\(^5\)This network can be accessed at http://diseasome.eu/
3 HuDiNe

Hidalgo et al. identified pairwise comorbidity correlations of more than 10,000 diseases based on the clinical records. This includes the comparison of diseases with different prevalences\(^6\). Therefore, they calculated the comorbidity with two statistical measures: the Pearson correlation \(\phi\) and the Relative Risk \(RR\). Both measures tend to over- or underestimate the comorbidity under certain circumstances [HBBC09].

3.1 The data

The datasets are provided as ASCII text files ranging from a size of 20 MB up to 600 MB. There are 18 datasets overall, 9 with 3 digit ICD-9 diagnoses and 9 with 5 digit ICD-9 diagnoses\(^7\). The datasets are further divided by sex and race. Each file holds 10 tab-separated columns as shown in Table 1.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>D2</td>
<td>P1</td>
<td>P2</td>
<td>Co-occurrence of D1 and D2</td>
<td>(RR)</td>
<td>(99%) Conf. Interval (left)</td>
<td>(RR)</td>
<td>(99%) Conf. Interval (right)</td>
<td>(\phi)</td>
</tr>
</tbody>
</table>

Table 1: Columns of the data. D stands for the ICD-9 code of the diseases, P for their prevalence. Note that both the prevalence and co-occurrence are the absolute number of affected patients from the specific population.

3.2 The graph

The visualization is done as follows: Each node in the graph represents a disease, more precisely an ICD-9 diagnosis. If there is a comorbid interaction with another disease found in the dataset, both nodes are linked with an edge. Both, the size of the nodes and the thicknesses of edges are visualized according to the prevalence of the diseases and the strength of the comorbid relationship. More prevalent diseases are represented by larger nodes, and stronger comorbidities are visualized by thicker edges between these nodes.

In the ICD classification diseases are assigned to disease categories based on the first 3 digits of a given ICD-9 code. This classification is represented by a specific coloring of the nodes.

Because neither the short nor the long description of the disease is part of the HuDiNe datasets, a mapping between ICD-9 diagnoses and disease names, given in a short and a

---

\(^6\)The Prevalence of a disease is the proportion of individuals in a given population diagnosed with this disease. However, in the HuDiNe datasets it is the absolute number of diagnosed patients.

\(^7\)International Statistical Classification of Diseases and Related Health Problems Version 9 in the Clinical Modification. See http://www.icd9data.com/
long description, is automatically loaded in the database by CoVi upon initialization. This not only allows CoVi to label the nodes in the graph with the corresponding disease names, but also makes it possible to search for diseases by their name.

To cluster a given graph, we use the Borderflow algorithm which performs well on large graphs [NS09]. The basic idea behind Borderflow is to maximize the flow within a cluster C and to minimize the flow from the nodes within C to the outside of it. Here the flow is the strength of the comorbid relationship between diseases. So Borderflow will divide a given graph into clusters, such that the comorbid relationships between diseases in a cluster are maximized, while minimizing comorbid connections of the diseases of a cluster to diseases outside of it. As a result, Borderflow divides a disease network into groups of diseases with a high comorbid relationship between them. In the current version we use Borderflow to compute a soft clustering, which means a disease can belong to more than one cluster.

4 CoVi

4.1 Application

The main challenge for our application is to handle large datasets in forms of flat files with up to 600 MB. In order to support a fast search, CoVi is backed by a MySQL database. Therefore, the user needs to specify a database connection with administration permissions on start-up. If such a database connection exists, we support a fast loading of HuDiNe datasets into this database. The database schema is rather straightforward. For each dataset we create a database table with the same 10 columns as in the HuDiNe files. In order to find all comorbid diseases to a given disease \( d \), we will have to find all rows where the ICD-9 code of \( d \) occurs in the first or second column. Once the user selects a dataset for exploration he has access to the main user interface of CoVi. Figure 1 shows the user interface with a small-sized graph already displayed. The window is separated into four major parts. The top of the window is the toolbar controlling the main features of the application. The left frame allows users among other things to search for specific diseases in the dataset and to cluster the shown graph. The upper right side is the graph pane. Below the graph the edge table is shown. This table contains all information of the original datasets used to display the graph above.

Apart from that, CoVi has a variety of other features.

- CoVi is a platform independent tool with relatively few requirements: A Java Runtime Environment and a MySQL database accessible with Database privileges.

- Multiple datasets are supported, since for each dataset a database table is created. The loading and indexing is done rather fast. We use the optimized MySQL LOAD DATA command and create indices over both ICD Code columns.

- All four graph Layouts as HuDiNe: Spring, Hierarchical and Radial-Hierarchical
as seen in Figure 1, and the Circle layout\(^8\). The user can furthermore control the visualization, such as stop animations (in the Spring layout), drag and drop nodes, or zoom in and out.

- A soft clustering of the graph using the Borderflow algorithm. The clusters are also represented in the graph through specific coloring.

- All nodes can be labeled with the ICD-9 code and the disease name if it exists: Upon startup, CoVi automatically creates a database table mapping ICD-9 codes to disease names.

- A search for diseases, either by ICD-9 code or an approximate search using disease names. Users can specify, whether RR or \( \phi \) is used to find comorbid diseases. Moreover, users can set minimum thresholds for these values, so CoVi looks for comorbidities only beyond this value.

- An exploration of the network by double clicking a node in the graph. A search is performed on this node using the specified measure and threshold. The new found comorbid diseases are added to the network.

- Additional features, such as the export of the current network as a picture or its data as a CSV file, switching between available datasets in the database, generating hyperlinks to Wikipedia and DBPedia are based on the name of the selected disease.

\(^8\)See [http://pcai042.informatik.uni-leipzig.de/~swp10-8/] for more screenshots
4.2 Architecture

4.2.1 Design

For the task of creating an interactive graphical representation we used the prefuse Information Visualization toolkit\(^9\). The prefuse toolkit is designed according to the *Information Visualization Reference Model (IVRM)*. The IVRM divides the information visualization task into discrete steps: First, the source data is acquired and transformed to internal structures. Second, these data structures are modeled to a visual encoding, e.g. by defining shapes and colors of the visual items. Finally, the data is presented and the displayed data can be modified by user interactions. The prefuse library is designed and partitioned accordingly into the major packages *prefuse.data*, *prefuse.action*, *prefuse.visualization*, and *prefuse.display*. Each package handles certain data structures and transformation tasks.

The *HuDiNe* datasets are provided as flat files but our application uses a MySQL database as backend for higher performance. Thus, our main task was to develop a database scheme to hold the source data and to support a transformation into prefuse tables, a prefuse data structure in the *prefuse.data* package. Furthermore, to support our custom graph design, which depends on the source data properties, we had to implement our own prefuse visualization classes. Those instances hold the needed data to actually paint the graph. We encapsulated each graph design property, e.g. shape, size and color of the graphical items, in separate classes which are integrated in the mentioned visualization classes. Therefore, we support convenient customization of the graph design. Finally, we had to define the interactions between the user interface and our data backend.

We will now give a brief overview of the architecture of our application. According to Figure 2, CoVi is separated into the following five packages analogously to the prefuse toolkit. The package structure is based on hierarchical layers according to the IVRM.

---

\(^9\)http://prefuse.org/
db_connector constitutes an interface between the data backend of the MySQL database and the application. The main tasks of this package are to manage the database connection, build and execute queries for network explorations, and search for diseases by their ICD-9 code or name, which is the initial step for every exploration of the network.

In order to have a valid connection to our data backend, we need not only to establish a connection to the MySQL database server, but we also need to create and maintain metadata which allows using more than one dataset. Furthermore, we created a table to map ICD-9 codes (both 3 and 5 digit codes) to the names of diseases (more precisely a short and long description of the disease). The DatabaseReader class handles exploration requests from the GraphManager, the main class of the data package, and retrieves the necessary data from the database. This is done by preparing a SQL query for the desired disease to explore the network with regard to the chosen threshold. Internally, it uses the DataBaseDataSource instance to alleviate the data transformation process from a JDBC ResultSet to our custom data format: a prefuse Table instance.

data serves for the internal representation of the raw data. This handles the transformation into prefuse data structures, initializes the Borderflow algorithm with the current graph data, and saves the results into custom data structures.

The main class of this package is the GraphManager. It manages the tables, which hold the data of edges and nodes with their specific properties of the source data, e.g. prevalence, ICD-9 Code, φ etc. It communicates with classes of the db_connector package and the visual_abstraction via the Observer Pattern. Therefore, it is possible to send requests to the classes of the db_connector package for an exploration and to request the classes of the visual_abstraction to update the visualization once the graph is explored.

The second task of this package is clustering the disease network based on the edge weight values φ or RR, which are saved in the edgeTable of the GraphManager class. We implemented an extra class for clustering to avoid a mix of graph operations like exploration and clustering. This class uses a modified version of the Borderflow algorithm, which allows prefuse tables to be used for input as well as output. The Cluster class is also connected by the Observer pattern with the GraphManager class, to react to deletions or explorations of the current graph. Therefore, the data used to cluster the network is equivalent to the current graph. Furthermore, this class is observable, so the CVVisualization class can react, if the current graph is clustered and then updates the visualization accordingly.

visual_abstraction holds classes namely the CVVisualization class, in order to enrich the data with all necessary information for visualization. The action event registration is also done here. The user is thereby enabled to interact with the graph by calling those registered actions.

view and controls provides classes to create the graphical user interface (GUI) and handles user input, though some controllers, e.g. for the buttons, are defined as anonymous inner classes and are not part of the controls package. This is a convenient way to make the code easier to read and a replacement of those GUI items can be done rather fast.
4.2.2 Event handling and communication

We handle all database actions using classes and methods in the `db_connector` package to have a better control over the dynamics of the application. We defined methods for all database related tasks, which basically wrap around the according JDBC methods in most cases, but also change the internal state of the application, allowing us to update the GUI.

The event handling of CoVi is a workflow orientated MVC Pattern. Each function has its own controller and view component. This provides a simple change or extension of existing functions with respect to the IVRM Model, which separates the model in source data and visualization data, there exists a strong interdependence between the classes of the `data` package and the `visual_abstraction` package. By using both `Observer Pattern` and `PropertyChangeListener`, we ensure that the visualization data is synchronized with the graph data. These mechanisms are also responsible for synchronizing other GUI components, such as the tables listing the data behind the presented graph with the current graph.

The event dispatching follows the subsequent hierarchical strategy: Each class communicates with its neighbor classes in the layer structure, thus assuring a class only handles responses holding data structures of neighboring layers. Figure 3 shows an example.

To make CoVi as modular as possible we defined the `GraphExplorer` interface which defines the central method for reading the raw data. The goal was to make most of the code of CoVi reusable for future versions, that will probably work without a database as backend.

5 Evaluation

As the dataset could be rather large, we have tested CoVi. For a good usability two main questions are most relevant:

1. Loading performance: How long does it take to load datasets into the database?

2. Exploration performance: What time is required to perform a search for a disease?
   In more detail: How long does the actual search in the database take and how much time is required for CoVi to display the results as a graph?
All tasks were tested on an Intel 1.83 GHz Dual Core machine with 2 GB RAM running a Java Runtime Environment Version 1.6.026 and a MySQL database Version 5.1.41 under Windows 7.

5.1 Loading performance

To address the first question, we have loaded the largest HuDiNe dataset AllNet5 into a newly created database. The size of this dataset is 677 MB and thereby the largest in the HuDiNe collection. It holds information about all patients Hidalgo et al. collected from medical records in great detail [HBBC09]. In total, 13,039,018 patients were analyzed and 6,088,553 comorbid interactions of diseases were identified. The diseases are specified by 5 digit ICD-9 codes.

To create a table in the database and load all information into it, CoVi required 43.363 seconds, indexing the first row took 91.62 seconds, and indexing the second row took another 190.79 seconds. In total the task of loading the AllNet5 dataset took 5 minutes and 26 seconds.

This fast loading performance (12.4% of the total time) is possible due to the optimized MySQL LOAD DATA INFILE command, rather than reading files manually. As our main focus is to support a fast search and exploration of the dataset, we intentionally create indices over both rows identifying the diseases. This sums up to approximately 70% of the total time needed to load the dataset and make it available for exploration in CoVi. However, the default use case is to load a dataset into the database only once. Therefore, we think the additional time to create the indices is appropriate, in order to provide a rather fast exploration performance.

5.2 Exploration performance

An exploration of a chosen dataset is performed by using the Disease Search Panel or by double clicking a node in the graph. Either way, it is a search in the dataset for comorbid interactions with this node, limited by the chosen threshold on either of the measures $\phi$ or RR. For CoVi this task can be divided into two main parts: query the database and process the results in order to visualize them. Figure 4 shows the results of some example explorations in the WhiteMaleNet3 dataset from HuDiNe.

As to be expected for explorations with more comorbid diseases in the result set (by using lower thresholds), CoVi requires more time to process the data. This is due to the fact that a prefuse graph relies on two data structures: a node table and an edge table which are handled by the GraphManager in the data package. Therefore, CoVi first needs to process the result set of the SQL query in order to identify all nodes and check, whether they are already visualized in the current graph and therefore already in the node table. Accordingly, larger exploration tasks increase the amount of time CoVi requires to process the data before visualizing it.
As shown in Figure 4, exploring diseases with up to 500 connections to comorbid diseases does not take more than 2 seconds. A rule of thumb is that CoVi performs better on large exploration tasks if the user has chosen a graph layout that is not constantly animated like the Spring layout. Then, even large networks with thousands of nodes could be visualized, with more time needed, to do the exploration. Such explorations would be rather rare, because visualizations of such large networks will destroy any advantage of visualization compared to a statistical analysis of the datasets.

6 Conclusion and discussion

We have developed an easy-to-use application to display and explore Human Disease Networks. Like HuDiNe, CoVi supports various network layouts. Additionally, it gives the user more control to create a customized network layout. Moreover, it is possible to cluster a network using the Borderflow algorithm. For future versions of CoVi, a more generic approach to clustering would be appropriate. Thus, let the user decide, whether to compute hard or soft clusters. Users could even benefit from an interface for other clustering mechanisms, such as k-means or canopy clustering.

Our main focus was to develop a real-time graph visualization and exploration tool for the HuDiNe datasets. Therefore, we chose a database as data backend. Although we defined the DataReader interface as a bridge between the data and db_connector package, a development of a version not depending on a database would require changes in the view and control packages as well.

Nonetheless, CoVi is of great help to analyze Human Disease Networks and it is particularly of great interest not only to medical experts, but patients as well. As of now CoVi
only supports the visualization of data conforming to the *HuDiNe* format. However, it would be possible to use the core of CoVi for other network visualization tasks as well.

**Acknowledgments**

We developed CoVi as part of the course *Softwaretechnik Praktikum* during the summer semester 2010 at the University of Leipzig. Besides the authors of this paper, Moritz Kähler and Marcello Scrobanita were also part of the development team. We want to thank Dr. Axel C. Ngonga Ngomo and Patrick Fomy for their continuous input and great support.

**References**


