Network Analysis and Visualization
Identifying most prominent nodes

MSc student ID: 1402590
A. M'Tir

Supervisor: Dr. Shan He

Submitted in conformance with the requirements
for the Master degree in computer Science
Stream 1
School of Computer Science
at the University Of Birmingham
in United-Kingdom

September 2014, School of Computer Science, University of Birmingham

Abstract

The aim of this thesis is to analyze and visualize different networks, ranging from social networks to biology networks. More precisely, we focused on tackling and addressing the problem of identifying most prominent nodes in different medium scale networks. Three highly technical and complex detection algorithms have been implemented, namely the k-cores algorithm, the bias random walk and finally the core-periphery profile. (Research project)
These algorithms have been evaluated, compared and contrasted to a baseline algorithm using four data sets where the prominents nodes are known.

To reach this goal, I devised a web application, a client server application, offering an online service to users interested in visualizing, analysing and characterizing their networks (Software project).
This software framework have been developed using the latest technology such as jQuery, java-script, HTML5, CSS3, AJAX, JSON, Java EE API, servlet, JSP and Tomcat server. The originality of this work is that it offers an online service, a web application to users who want to analyse their networks to determine the most important vertices.

Keywords

Graphs, Networks, Prominent nodes, visualization, metrics, algorithms, K-Core decomposition, Biased Random walk, Core-Periphery Profile, Web-application, client-server, Java, JSP, Servlet, Tomcat7, JSON, AJAX, jQuery.

Declaration

I declare that the report has not been published in any journals, websites and magazines. Also, it has not been appeared in the University of birmingham as well as other universities. All the related works has been finished supervised by Dr. Shan.He

Acknowledgement

I would like to express my very great appreciation to my wife Rita for her patience, support, assistance and trust.

Table of Contents

Chapter 1: Introduction and background

1.1 Networks and graphs

A network is the outcome of interactions between a large number of entities such as humans, businesses, molecules, cells, proteins, and stocks. A network is represented by a graph: a set of nodes (vertices) connected by links (edges).

The Figure 1 shows a social network of 15 office workers and their associations. The nodes represent the office workers. The edges between nodes represent the association and frequent communication. This social network has been reported by Thurman (1979). He spent 16 months observing the interactions among employees in the overseas office of a large international corporation.

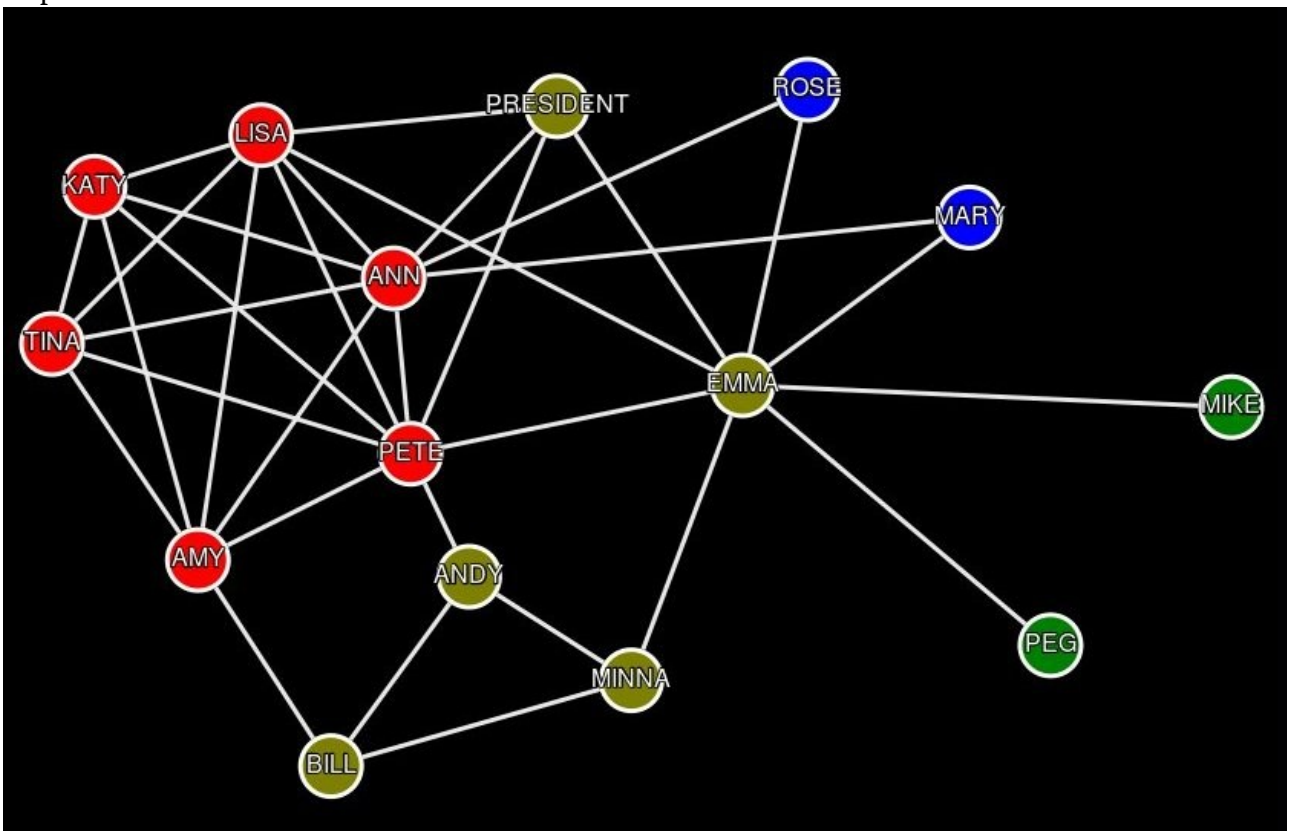


Figure 1: Example of Social Network, Thurman Office ((15 nodes, 33 edges) Thurman B. (1979). In the office: Networks and coalitions. Social Networks, 2, 47-63.)

Networks are important because they allow us to model, detect and extract significant patterns or structures and useful information about the system. They are the result of interactions between a large number of entities and represent a quite wide large number of systems. We represent a network in a graph form: a set of nodes connected by edges. These edges or links could be directed and weighted. In our project we have restricted our algorithms to the undirected and unweighted graphs networks.

1.3 Properties and characteristics of prominent nodes

The importance of nodes is a vague concept, and the measures to identify these nodes are based on different aspects of properties, and often conflict with each other.

In social network, the importance of nodes was discussed as “centrality”, a characteristic of a node's position in a network. Three measure indicators [11] are often suggested for node centrality, namely, the degree, betweenness, and closeness.

Intuitively, a prominent node should have a large number of edges (high degree) compared to a regular node. Indeed, in social networks, they have noticed that prominent nodes have on average 3 to 4 times as many friends as regular nodes. The degree of social networks often follow a power law: there are few nodes with an exceptionally high degree and many nodes with a lower degree. However, we may have nodes in the network with many connections, that are not prominent, or vice versa, prominent node with a smaller number of connections.

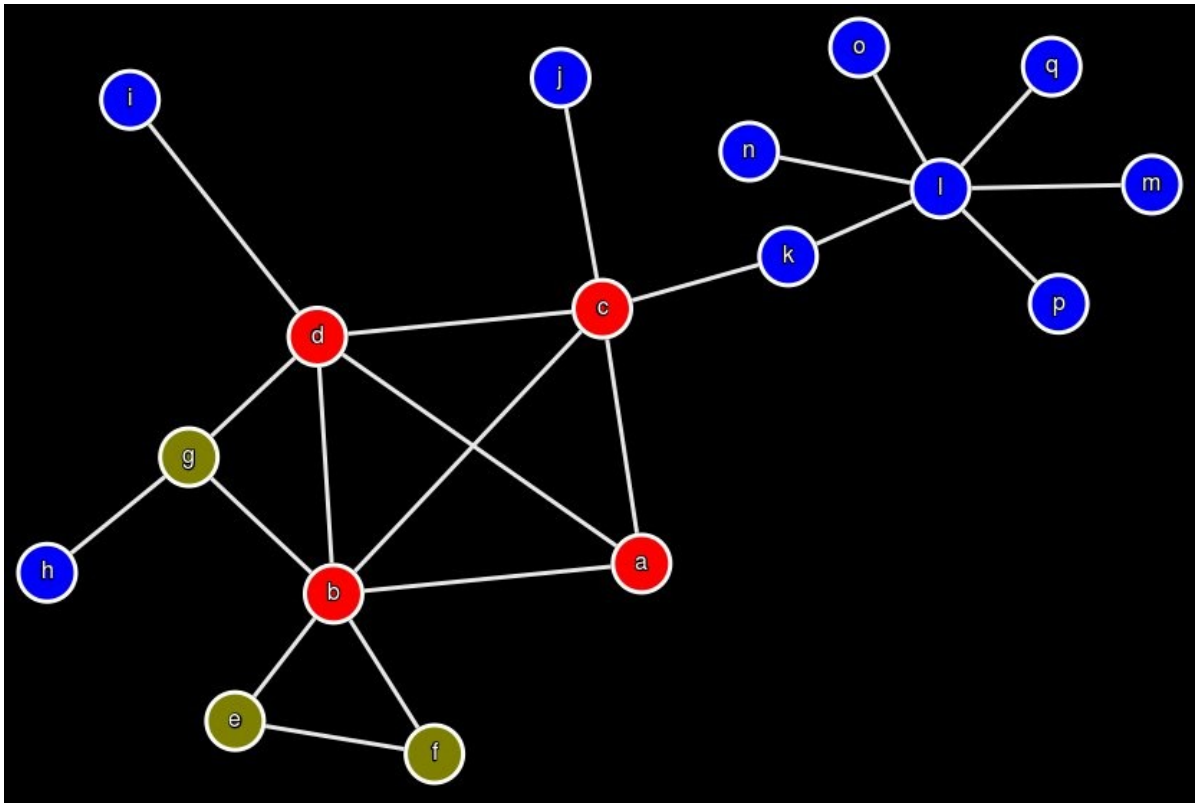


Figure 2: Example of node l with many edges on the periphery.

To illustrate this case, we can easily imagine a star network, on the periphery, connected to the rest of the network by just an edge. For example, in the Figure 2, the node l has six edges and is connected to the network by just one edge. Therefore the subgraph composed of the node l and its neighborhood is Isolated on the periphery and is not considered a prominent node.

In general, the high degree nodes function as a hubs, and are often grouped in a densely connected core, realizing the short pairwise distances (betweenness) between the more peripheral nodes. Therefore, we expect that the degree and betweenness measure of a node to play an important role in determining these prominent nodes.

Finally, a third important characteristic of a prominent node often mentioned, is the closeness centrality which emphasizes the distance of a node to all other in the network. The closer one is to others in the network, the more favored is that node's position.

1.4 Aims and objectives

The aim of this master project is to devise a software tool to visualize and analyze networks, and more precisely to tackle and address the problem of identifying most prominent nodes in different networks. Therefore the application should offer to users a means to visualize and gather different statistics about their network. Secondly, the application should offer different algorithms to detect prominent nodes.

1.5 Project type

The project type is a combination of *Software* and *Research project*. The *software project* part consists of devising a reliable robust online web-application using client-server architecture.
On the other hand, the *research project* part implies implementing and using different algorithms to extract statistics and detect prominent nodes in the network.

Chapter 2: Methodology and Algorithms

2.1 Methodology

We will consider various existing algorithms, different methods of determining the importance of a node in a network. We will apply these algorithms to four different networks where the prominent nodes are known. Then we will analyze the performance and precision of these detection methods.
Also we will compare these algorithm to a simple baseline algorithm based on the closeness and degree measure to quantify any added value.

2.2 Algorithms

2.2.1 Network Statistics

We offer to users a means to gather and collect statistics about their network prior to apply any algorithm. We have designed a specific tab where users can upload their networks and gather information such as:

Network size: Network graph G(V,E)

Number of Nodes: $|V|=n$, Number of Edges: $|E|=e$, Density: $\frac{n*(n-1)}{2}$ Maximum number of ties, links or edges for an undirected graph.

Characteristic path length: The average number of steps along the shortest paths for all possible pairs of network nodes. It measures the *efficiency of information* or *mass transport* on a network. Concretely, it measures for instance the number of people you will have to communicate on average, to contact a complete stranger, or the average number of clicks which will lead you from one website to another.

$$l_G=\frac{2}{n*(n-1)}\sum_{i\neq j} d(v_i,v_j) \qquad \text{(Equation 1)}$$

where $v_i, v_j \in V$ and $d(v_i, v_j)$ is the shortest distance between v_i and v_j

Diameter: the longest shortest path between any two nodes in the network.

$$d = \max_{v \in V} \epsilon(v) \qquad \text{(Equation 2)}$$

Where $\epsilon(v)$ is the eccentricity of a vertex v, the greatest shortest distance between v and any other vertex.

Degrees' distribution:
The degree distribution is very important in studying both real network, such as the internet and social networks, and theoretical networks. The degree distribution is the probability distribution of the degrees, number of connections, over the whole network.

Centrality is a characteristic of a node's position in a network. It is a micro-level measure.
Degree centrality of a vertex is the number of connections a node has. This indicator does not take into account indirect ties an entity has.

Closeness centrality emphasizes the distance of a node to all other in the network. The closer one is to others in the network, the more favored is that node's position.
Farness is the sum of the shortest (geodesic) distances from a particular node to all others in the network. Closeness is the reciprocal of farness.

$$Closeness(v_i) = \frac{1}{Farness(v_i)} = \frac{1}{\sum_{i \neq j}^{|V|} d(i,j)} \qquad \text{(Equation 3)}$$

Degree centrality measures one's local position, while **closeness centrality** measures position globally.

Betweenness centrality is the extent to which a node falls on the geodesic paths between other pairs of nodes in the network. The more nodes depend on a node to make connections with others, the more power that node has.

Centralization: ia a measure at the macro level. There are as many centralization measures as centrality measures. It indicates how unequal the distribution of centrality is in a network or how much variance there is in the distribution of centrality network. Centralization is a characteristic of a network.

Centrality measures have been popularized by social scientist in the 1970's as possible measures for the importance, or prestige as they call it in a social network [11]. Such measures assume that an entity that is central in the network, meaning that it is connected to a lot of other entities via some short path, is prominent.

By far the simplest and most common measure in the case of social network, is the degree centrality. As we have seen previously, the number of edges is a good indicator but definitely not perfect. In the baseline algorithm, we use the closeness and degree measure to select the l most prominent nodes in a network. First we sort them by closeness measure and in case where some entities have the same value then we sort them by degree. This baseline algorithm will be compared to the other algorithms to quantify any added value.

2.2.3 K-Core decomposition

K-Core (or core of order k) decomposition is a widely used method aimed at partitioning a network graph G=(V,E) in layers, from the external to the more central ones. K-cores are subsets of a network whose cohesion increases as k increases. For small values of k, the k-cores tend to be large, diminishing in size as k increases. Therefore the k-cores are used to define the core collapse sequence of a network.

To fully understand the k-core decomposition, it is worth to introduce some important definitions and concepts.
A subgraph H=(C,E|C) induced by the set $C \subseteq V$ is a ***k-core*** if and only if the degree of every node $v \in C$ induced in H is greater or equal than k ($\forall v \in C: \ degree_H(v) \geqslant k$), and H is the maximum subgraph with this property. A vertex i has **shell index k** if it belongs to the k-core but not to (k+1)-core. A ***k-shell*** S_k is composed by all the vertices whose shell index is k. The maximum value k such that S_k is not empty is denoted kmax. The k-core is thus the union of all shells S_c with $c \geqslant k$. Each connected set of vertices having the same shell index c is a cluster Q^c , where the corresponding set of edges are those connecting vertices of the cluster. Each shell S_c is thus composed by ***clusters*** Q_m^c , such that $S_c = U_{1 \leqslant m \leqslant Q_{max}^c} Q_m^c$, where q_{max}^c is the number of clusters in S_c . The k-core decomposition therefore identifies progressively internal cores and decomposes the networks layer by layer, revealing the structure of the different k-shells from the outmost one to the most internal one, as shown bellow with our software framework in the Figure 3.

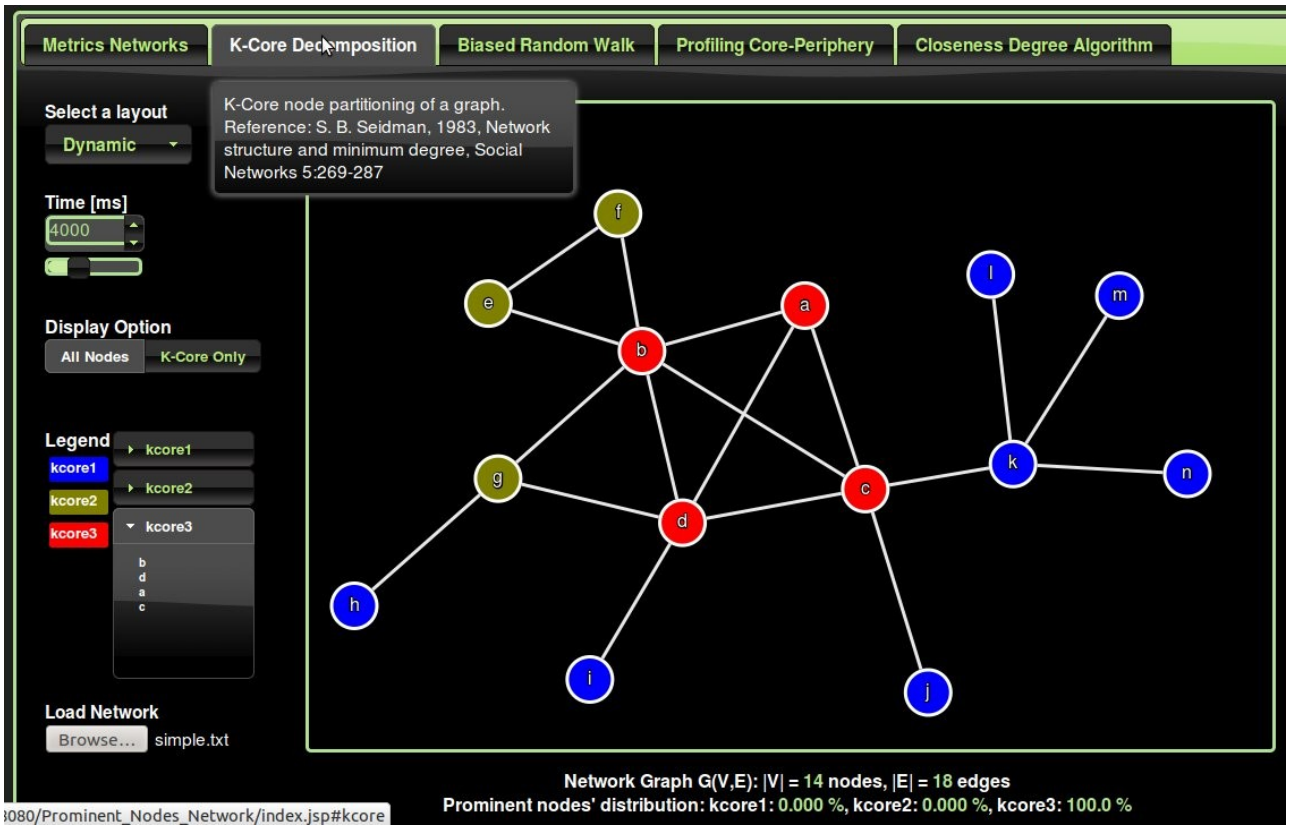


Figure 3: Sketch of the k-core decomposition for a small graph. The color on the vertices distinguish different k-shells.

To sum up, the k-core decomposition consists in identifying particular subsets of the network,

called k-cores, each one obtained by a recursive pruning strategy. In other words, the k-core is obtained by recursively removing all the vertices of degree less than k, until all vertices in the remaining graph have at least degree k. The k-core decomposition therefore provides a probe to study the hierarchical properties of large scale networks, focusing on the network's regions of increasing centrality and connectedness properties.

2.2.4 Identifying prominent nodes using biased random walks

The classic Random Walk Algorithm generally traverses the graph, moving to a random neighbor with probability 1-p, and jumping to a random node with probability p.

The simplest intuition about prominent entities in a network is that they have a large number of connections. Therefore we expect the degree of a node to play an major role in determining the importance of a node.
A good first indication of importance could be the degree centrality:

$$f_{deg}(v) = 1 - \frac{1}{|N(v)|} \qquad \text{(Equation 1)}$$

However, as we stated previously, there may be some entities in the network with many connections, that are not important, and the inverse could also be true.

In their work Takes and Kosters [1] introduced the concept of a node's neighborhood density, based on the notion of triadic closure. The concept of triadic closure states that the majority of all friendships formed within a network takes place between two people who have at least one friend in common[8]. The probability increases with the number of common acquaintances as well as with the degree of a node. This phenomenon also known as a preferential attachment.

More generally, Takes and Kosters [1] argue that the friends of prominent nodes have more connections in common than regular nodes. They call this concept a node's neighborhood density (nd):

$$f_{nd}(v) = 1 - \sum_{w \in N(v)} \frac{|N(w) \cap N(v)|}{(|N(w) - 1|) * |N(v)|} \qquad \text{(Equation 2)}$$

The numerator in the equation 2 defines the number of common connections, whereas the denominator normalizes the result so that it is independent of the degree of node v or the degree of node w.

Frank Takes and Walter Kosters believe that a combination of the two measures above, may be able to efficiently identify the various prominent nodes. Therefore, given a current node v, they define the probability P(w) of selecting a node w in the neighborhood $w \in N(v)$ is equal to:

$$P(w) = \frac{\alpha f_{deg}(w) + (1-\alpha) f_{nd}(w)}{\sum_{u \in N(v)} \left(\alpha f_{deg}(u) + (1-\alpha) f_{nd}(u)\right)} \qquad \text{(Equation 3)}$$

The Biased Random Walk takes as input an unweighted graph G(V,E) and parameters N, p, and α, and outputs a scalar value for each node v in V in the graph, representing its importance. N represent the number of steps in the random walk algorithm, p is the jumping probability, and

$\alpha \in [0,1]$ is used as a weight to focus either on the degree centrality and the neighborhood density.

Setting the value of α to 1 is roughly the same result as degree centrality, whereas a value of 0 turned out to give 0% success. This is because the degree plays a significant role in identifying a node's prominence, and very low degree nodes can still get a high neighborhood density score.

In our experiments, we follow the advice given by Takes and Kosters [1] where they set $\alpha = 0.5$, and $p = 0.15$, and the number of iterations N should be significantly larger than the number of nodes n by at least a factor 10 or 100.

Pseudo-Code Biased Random Walks

1: Input: Graph G(V,E), N, p, α

2: Output f[ ], containing the importance value f[v] for each node $v \in V$

3: for $v \in V$ do

4: f[v] ← 0

5: end for

6: i ← 0

7: v ← RandomNodeFrom(V)

8: While (i < N) do

9: f[v] ← f[v] + 1/N

10: if(rand(0,1) > p) then

11: v ← BiasedSelectFrom(N(v), α)

12: else

13: v ← RandomNodeFrom(V)

14: end if

15: i ← i + 1

16: end While

17: return f[ ]

2.2.5 Profiling core periphery structure by random walkers

The core-periphery profile algorithm provides a topological portrait of a network graph G(V,E) where $|V|=n$ is the number of nodes, and $|E|=e$ is the number of edges.

A numerical indicator, the *coreness* α_k, *k=1,2,...,n,* is attributed to each node, which for an undirected network is defined by the algorithm bellow. This coreness value highlights the node rank and role, and provides both a global network portrait as well as an individual characterization of each node.

$$0=\alpha_1 \leqslant \alpha_2 \leqslant \ldots \leqslant \alpha_n = 1 \qquad \text{(Equation 1)}$$

This numerical indicator of core-periphery separation, is based on the fact that core nodes have large closeness centrality, in other words a small average distance from the rest of the network. Therefore, this profile indicator is particularly adapted to networks with dense cores and a sparsely connected periphery. In other words, this method is aimed at revealing whether there exists a central core through which most of the network flow passes. It quantifies to what extent an actual centralization exists. By grouping all nodes with coreness below a prescribed threshold, we obtain the periphery. Moreover, it can reveal the peculiar role of some specific nodes. The core-periphery paradigm models the network as the union of a dense core with a sparsely connected periphery, highlighting the role of each node on the basis of its topological position.

In their paper, Rossa, Dercole and Piccardi [4] have derived the core-periphery profile, by an ***iterative algorithm***, a standard random walk model which provides the information to effectively asses the profile.

For an undirected network, they have defined the strength of a node i as

$$\sigma_i = \sum_j w_{ij} \qquad \text{(Equation 2)}$$ where w_{ij} is the weight between nodes i and j.

The persistence probability as the fraction of weight emanating from the nodes of S remaining within S.

$$\alpha_S = \frac{\sum_{i,j \in S} w_{ij}}{\sum_{i \in S, j \in V} w_{ij}} \qquad \text{(Equation 3)}$$

<u>Pseudo-Code algorithm used to implement the algorithm:</u>

<u>Step 1:</u> Select a random node i among those with minimal strength.

<u>Step 2:</u> k = 2,3, …, n: Select the node that minimize the persistence probability.

If it is not unique, select at random one of the node with minimal strength σ_k among those attaining the minimum.

$$S_k = S_{k-1} \cup v_k \qquad \text{(Equation 4)}$$

Intuitively, we start from the least connected node because peripheral nodes have less connections than core nodes. Then we grow our periphery set S by adding one node at a time. We use the persistence probability to quantify this. The inclusion of the most connected nodes are typically left at the end.

Chapter 3: Requirements, design and implementation

3.1 Requirements definition

In this section, I will list the requirements for our software system and describe what the system should do, the services that it provides and the constraints. Small scenarios to describe the different functional and non-functional requirements. Prior to the use cases diagrams and prior to the use cases scenarios development phases.

Requirement 1: The Software application should offer an online-service in the form of a web application using browsers and client-server architecture. The application should simultaneously handle multiple users and be responsive and robust.

Requirement 2: The user should be able to create its own network using a simple text editor, in the .txt format. The format should be intuitive and easy to understand, close to the concept of edge-list graph notation. Something like follow: [Source_Node][Space_Character][Destination_Node] for each line in the document network file. The user should be able to enter any comments prefixing them by # character if necessary and specify the prominent nodes if they are known prefixing them by the @ character.

 # Network Graph
 # Simple Star Network (6 nodes, 5 edges)
node1 node2
node1 node3
node1 node4
node1 node5
node1 node6
@knownProminentNodes
node1

Requirement 3: The client should be able to visualize its network using different layouts, and gather statistics and metrics such as the size (number of nodes, edges), average path, density in percentage, diameter, centralization measures (degree, closeness, ...).These indicators should be grouped and presented cleanly in table with a label in front of each value. Also, the degree distribution of the network should be displayed in a bar char graphic coupled with a curve form to offer a global view and reveal any obvious hiding statistical distribution, such as Poisson, logarithmic, exponential.

Requirement 4:
Algorithms detection
The application should offer to the user different algorithms to detect patterns and structures and identify prominent nodes. If these prominent nodes in the network are known then the web application should list indicators about the performance and precision obtained by each algorithm such as the percentage of prominent nodes found.

Requirement 5:
The interaction with the application should be intuitive and easy to use. The user should find the same feel and touch as a desktop application while using a any simple browser.

3.1 Client-Server Web Architecture

Our ***web application*** is a collection of ***web components*** that work together to provide specific functionalities on a web. We use the Java EE Enterprise Edition built on the Java SE Standard Edition, and our web components are either a ***Servlet*** or a ***Java Server Page (JSP)*** page.

The web application and its constituent components are managed and executed inside the ***web container***, also called a ***servlet container***, which provides additional features to the web application such as security. We have chosen the ***Tomcat7 container*** which also serves the role of the ***web server***.

When our web server Tomcat7 gets a request for specific functionality that a particular web component (such as a servlet/bean or a JSP page) can provide, our web server forwards the request to the servlet container in which the web component resides. That is, all requests to our web component that is responsible for processing data and generating the dynamic content are mediated by the Tomcat7 servlet container, as shown in Figure 4.

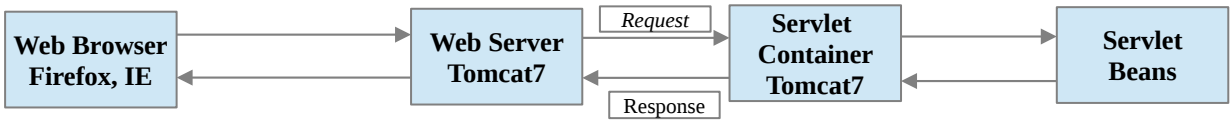


Figure 4: Request for a dynamic content.

The Java EE Servlet and JSP specification describe the service contract that a servlet container must provide and specify how a servlet should use those services. In terms of implementation, a servlet is a Java class that acts as a dynamic web resource.

When the client (web browser) makes a request, the web server (Tomcat) sees the resource path and determines that the resource requested by the user is not a static page and so forwards the request to the container (Tomcat).

The web container loads the corresponding servlet class and instantiates it. Only a single instance of the servlet is created, and concurrent requests to the servlet are executed on the same instance. Every client request generates a new pair of request and response objects. The container runs multiple threads to process multiple requests to a single instance of the servlet.

3.2 Model View Controller Pattern

The ***Model-View-Controller (MVC)*** architecture was developed in the 1970's in the smalltalk group. In our design we use the MVC architecture to clearly separate applications, both logically and physically, into three parts.

The ***Model*** is not only the data, but any enforced constraints on the data. Our model has been implemented as a collection of servlets/beans to perform different algorithm calculations.

The ***View*** part of the application that prepares and presents results to the user has been mainly accomplished by means of a JSP, JavaScript and CSS.

The ***Controller*** controls the interaction between the user and the application. The controller has been achieved by means of a special servlet, a subclass that extends extends the abstract HttpServlet

interface. This servlet accepts requests and implements the business logic.

The intent use of MVC pattern is to reduce the coupling among the three parts of an application, making our application easier to develop and maintain.

In an MVC Web application, a browser submits requests to the controller, which consults the model (which in turn stores and fetches the data and performs the calculation). Next, the model reports results to the controller and, indirectly, to the view. The controller then instructs the view to produce a result document, which is transmitted to the client for display.

To provide our web-application with rich user interfaces and responsiveness similar to those of desktop applications, we use the Ajax technology. Ajax is meant to significantly increase the speed of user requests that update only a small part of the displayed document. It does this by having the server provide only a relative small part of the document, the part that must change. This shortens the transmission time because the document being transmitted and rendered is much smaller. It provides great improvements in the richness of the web user experience, with applications that have frequent browser-server interactions.

The request from the browser to the server are asynchronous as shown in the Figure 5. This means that when a browser requests a new part of its displayed document from the server, it does not need to lock while it waits for the response. Both the user and the browser can continue to do something useful during the time it takes to fetch and render the new document part.

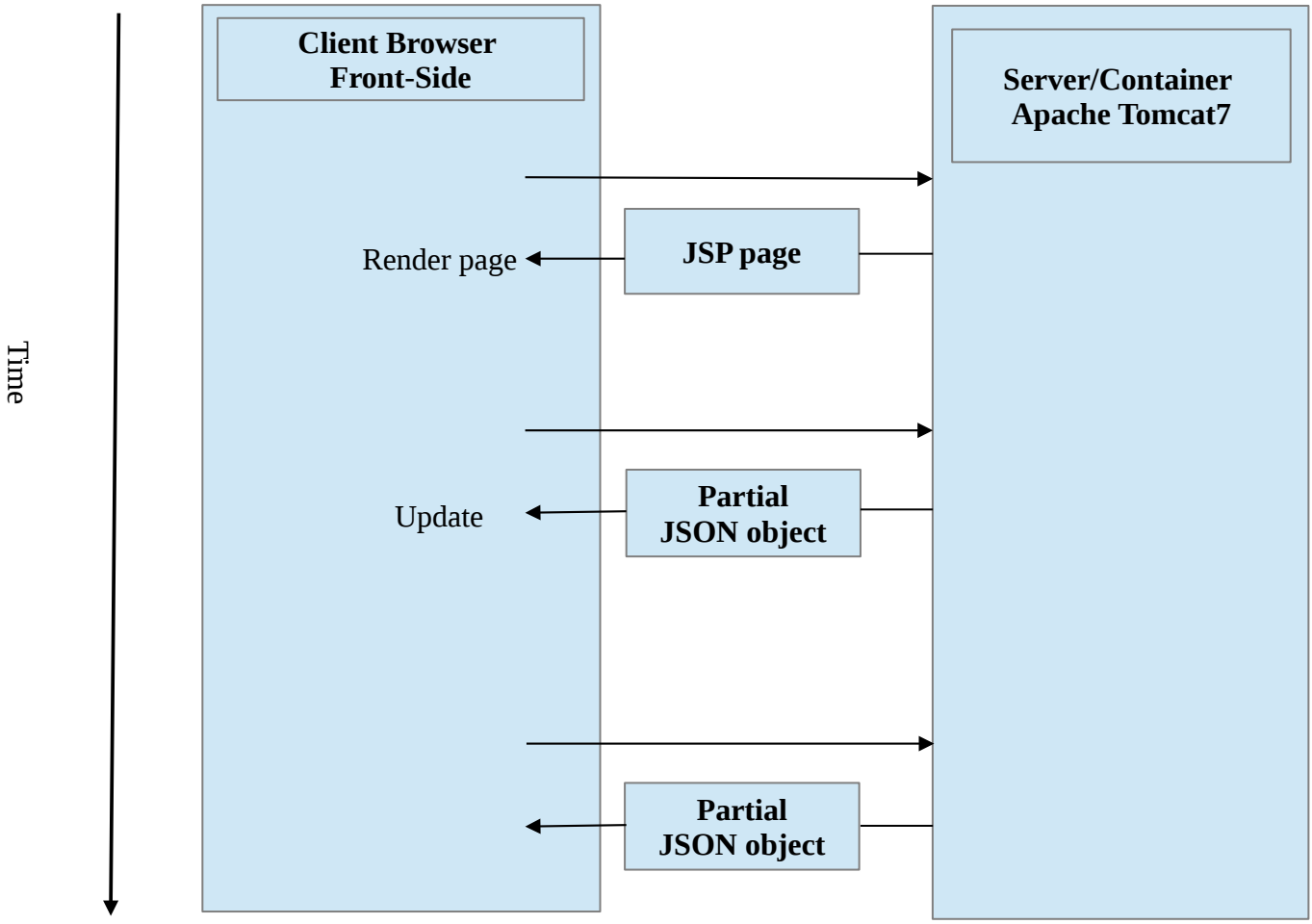


Figure 5: Ajax browser-server interactions

Through its name, Ajax uses JavaScript (JQuery) as its primary programming language. The x in Ajax represents XML. Usually, an Ajax request results in the server returning the requested data in an XML form. However other forms of data also are often returned. In our web-application, we exchange JSON object. JSON stands for JavaScript Object Notation. It is self-describing and easy to use. This notation is in a text format that can be read by any programming language.

To summarize, our design has one main JSP page (620 lines) with JavaScript (1800 lines) and CSS (140 lines) which implement the View part in the MVC model. The view (JSP) is composed of five tabs, where each tab offer a functionality. This main page is fully loaded the first time a request is made to the server and then partially updated for each request using the AJAX technology and JSON objects. This makes the web application more responsive, similar to those of desktop applications as the request is handled asynchronously and only few parts of the page are updated.

Each algorithm is grouped in a separate package with its own servlet(s). Therefore we ended with 6 packages and 8 servlets, namely,

Controller (1 servlet 200 lines) } (Controller in the MVC)

Metrics and Statistics (1 servlet, 230 lines)
Closeness and Degree Algorithm (1 servlet, 300 lines)
K-Core Algorithm (3 servlets, 400 lines) } (Model in the MVC)
BRW Biased Random Walk Algorithm (1 servlet, 500 lines)
PCP Profile Core Periphery Algorithm (1 servlet, 650 lines)

The class Controller is indeed a very special servlet as it implements the Controller in our MVC model. This subclass extends the abstract HttpServlet interface and therefore overrides the doPost method, to handle HTTP POST requests. The HTTP POST method allows the client to send data of unlimited length to the web server a single time.

Basically, the Controller class receives the user request data, in a JSON object form, identifies the request nature, forwards it to the adequate algorithm (Bean(s)) to perform the calculation, retrieve the results and then sends them back to the user.

See Appendix 2, for more details about the UML design and implementation.

Chapter 4: Validation and verification

Testing can only show the presence of errors, not their absence. [Edsger Dijkstra]

During the whole development, I adopted the Test-Driven Development (TDD) approach to develop the program in which I interleave testing and code development. Essentially, I develop the code incrementally, along with a test for that increment. I do not move to the next increment until the code that I have developed passes its test. Once all tests run successfully, then I move on implementing the next chunk of functionality.

These development testing phases are carried out at different levels of granularity. For instance, if I code a new feature, then I start by carrying out a Unit testing of that feature (class, method).
Now this new feature added belongs to a component, therefore it is important to carry out a component testing, where I focus on testing component interfaces. Finally, the system as whole is tested (system testing), to ensure that some or all the components in the system are integrated and interact properly.

Also whenever possible, I involved users to test the software system to gather input and advice on the system. It is practically impossible for me as a developer to replicate the system's working environment, as tests in the developer's environment are inevitably artificial.

During development, there are two major concerns that require constant testing as features are implemented and added to the initial application. The primer is to ensure that new features, algorithms will not potentially destabilize the initial software by behaving in an unexpected way (stability). The second is to make sure that the results returned are indeed correct and in the range and precision required and defined by the requirements (correctness).

4.1 Stability

Important to test thoroughly each function for unexpected inputs and boundary-cases, and anticipate the misbehavior of certain users. A number of these were achieved through the use of test-cases, included in appendix 1: Test Cases, JUnits.

4.2 Correctness

To ensure the algorithm return the correct result, deliberate input networks were used with the web-application, and the outcome was compared with the expected result. (See appendix 1)

Chapter 5: Algorithms' evaluation

Comparing with the Ground-Truth prominent nodes in a network G(V,E), where $|V|=v=n$ the number of nodes. This evaluation implies that we know exactly who are the prominent nodes in the datasets used. Let's assume that amongst the nodes in the network, there is a set $W \subseteq V$,of size $k=|W|$, which contains the nodes that are considered to be prominent. Naturally, $k \leqslant n$, k is a lot smaller than n, as only a small portion of the nodes is typically considered to be prominent.

In order to determine the importance of an algorithm, we sort the list of nodes by their importance value/criteria in descending order, and retrieve the subset I to be the top l nodes of the sorted list outputted by each algorithm.

Next, we define the following criteria:

Precision:

$$P=\frac{|W \cap I|}{I} \qquad \text{(Equation 1)}$$

Recall:

$$R=\frac{|W \cap I|}{W} \qquad \text{(Equation 2)}$$

F-measure:

$$F-measure=\frac{(2 \times precision \times recall)}{(precision+recall)} \qquad \text{(Equation 3)}$$

F-measure measures the balance between the two. Notice that if $l=k$, precision, recall and F-measure are equal.
Within the application we provide different networks datasets, (undirected and unweighted graph) varied in their nature and size. For some of these networks, the prominent nodes are exactly known, allowing us to verify and evaluate the precision and performance of the algorithms against this ground truth. The other datasets can be used to demonstrate the capability and pedagogy of the software tool developed.

We will start by describing and visualizing each of these data set, and if the prominent nodes are known, then a comparison of the algorithms are carried out.

5.1 First dataset: Zachary's karate club

Zachary's karate club: A social network of friendships between 34 members of a karate club at a US university, as described by Wayne Zachary in the 1970's. At the beginning of the study there was an incident conflict between the club president, John A., and Mr. Hi over the price of karate lessons. Mr. Hi, who wished to raise prices, claimed the authority to set his own lessons fees, since he was the instructor. John A., who wished to stabilize prices, claimed the authority to set the lesson fees since he was the club's chief administrator. As time passed the entire club became divided over this issue, which has led to a fission of the club as shown in the Figure 6.

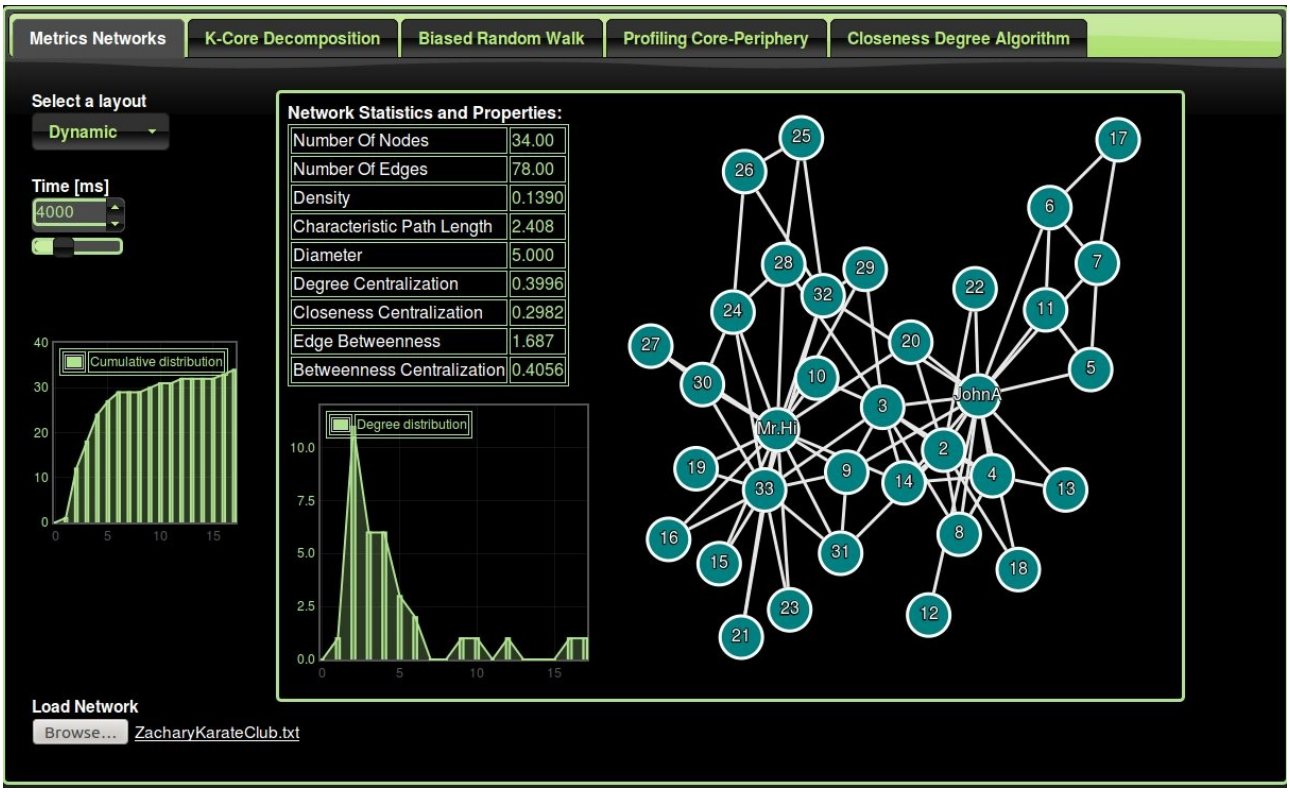


Figure 6: Network visualization and Statistics of the Zachary's karate club.

Knowing the important major prominent actor in each group, namely, Mr. John A. and Mr. Hi, we have used this dataset to evaluate the algorithms.

First, let's consider the K-Core algorithm. In the clustered view of the k-core algorithm in the Figure 7, we only show the cores and edges between them, The size of the core is proportional to the number of nodes that belong to it and the width of the edge is proportional to the number of edges between cores. We can notice that the two prominent nodes in the karate-club, Mr Hi and JohnA belong to the highest k-core.

Figure 8 is similar to Figure 7 as it represents the outcome of the K-Core algorithm applied to the Zachary Karate network but using the all node representation.

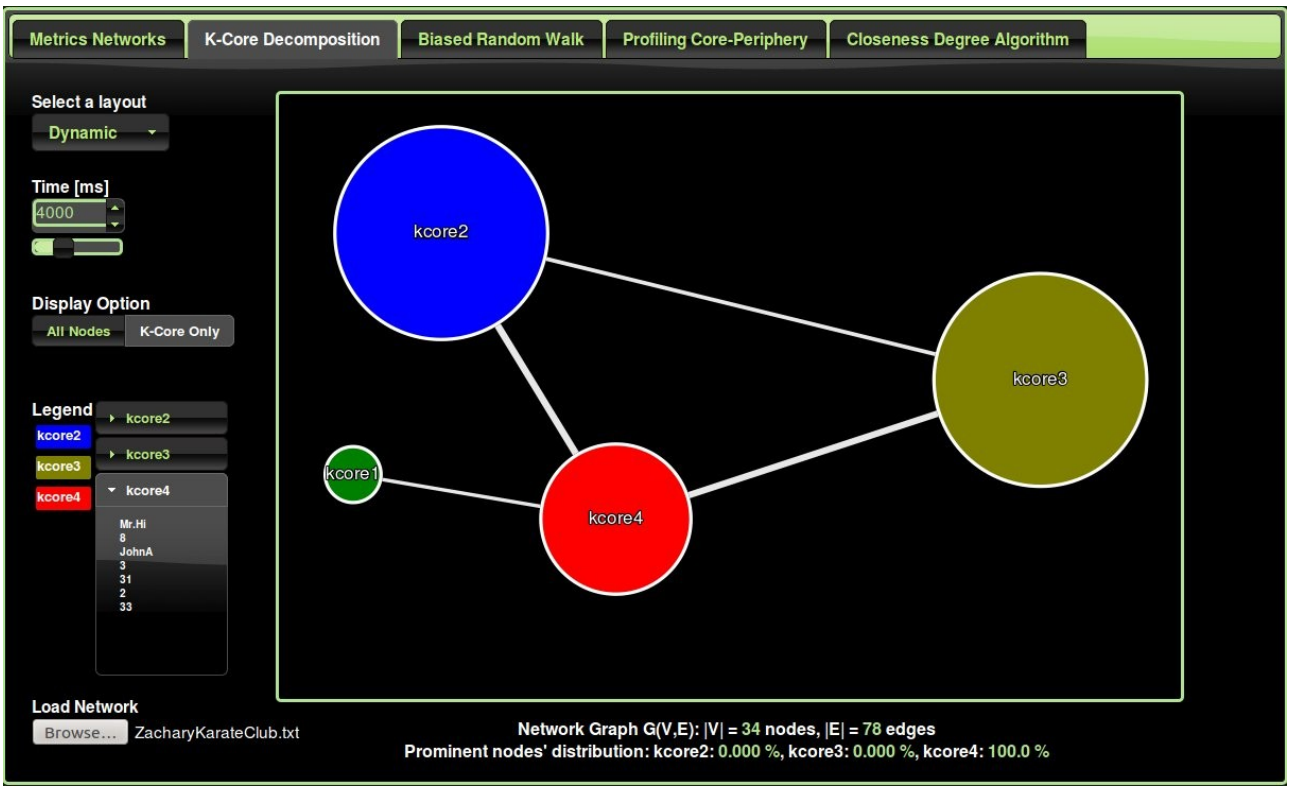


Figure 7: K-Core algorithm applied to the Zachary Karate club network (Core representation)

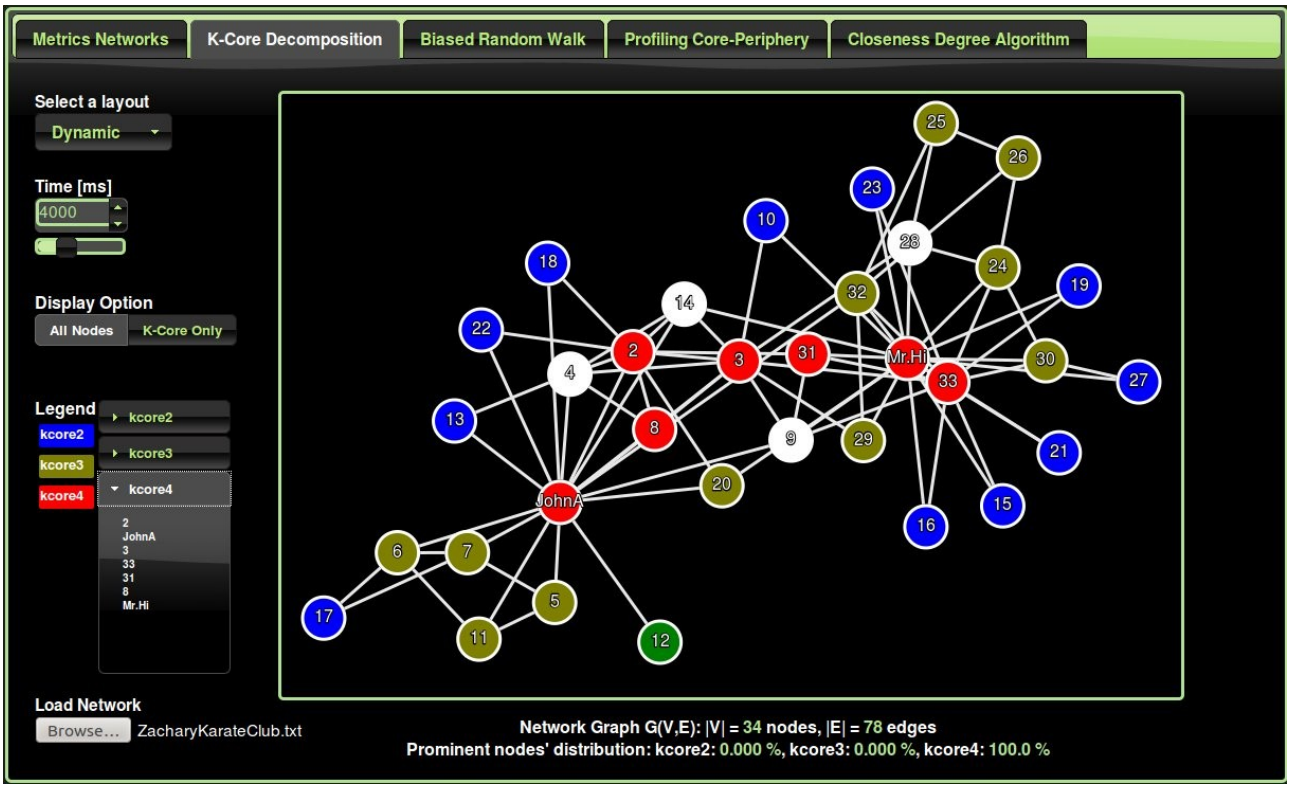


Figure 8: K-Core algorithm applied to the Zachary Karate club network (All nodes representation)

The second algorithm applied is the Profiling Core-Periphery algorithm. We retrieve the two most prominent nodes and compare them to the ground-truth. Both are indeed correct (Precision 100%).

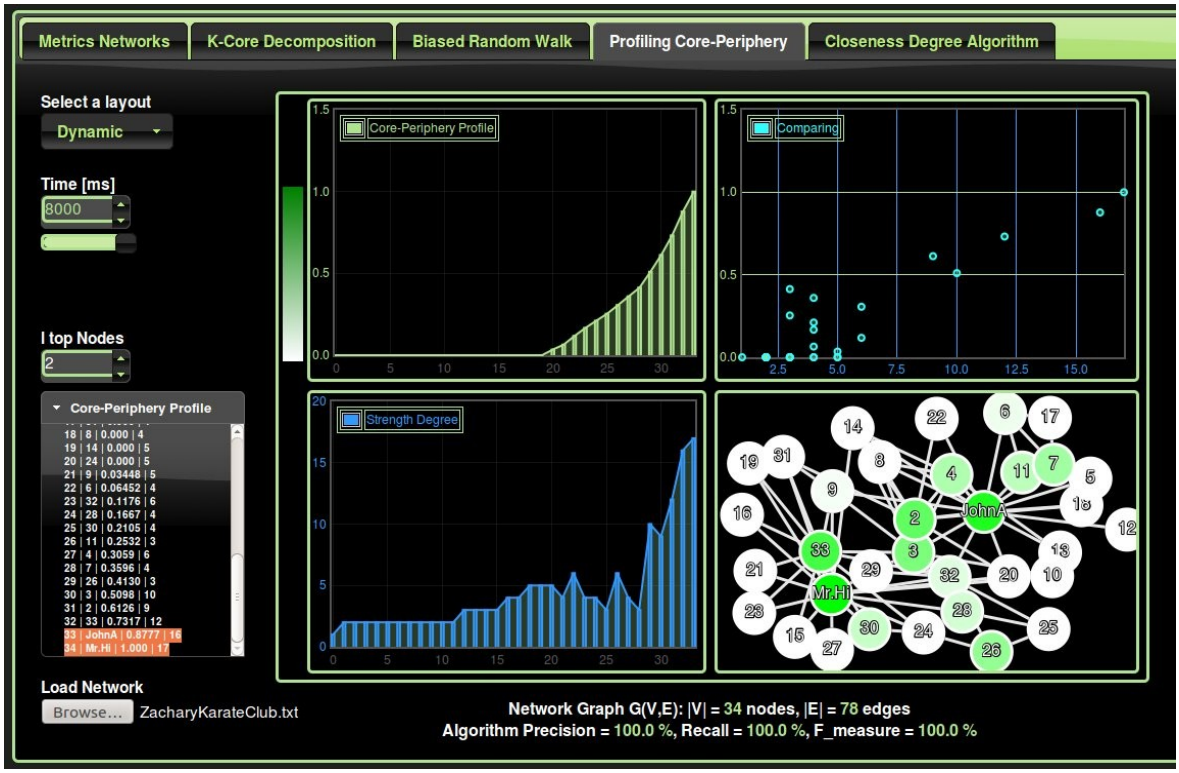


Figure 9: Profiling Core-Periphery algorithm applied to the Zachary Karate club network.

The third algorithm applied is the Biased Random walk algorithm (Figure 10). Unfortunately, only one is correct (Precision 50%).

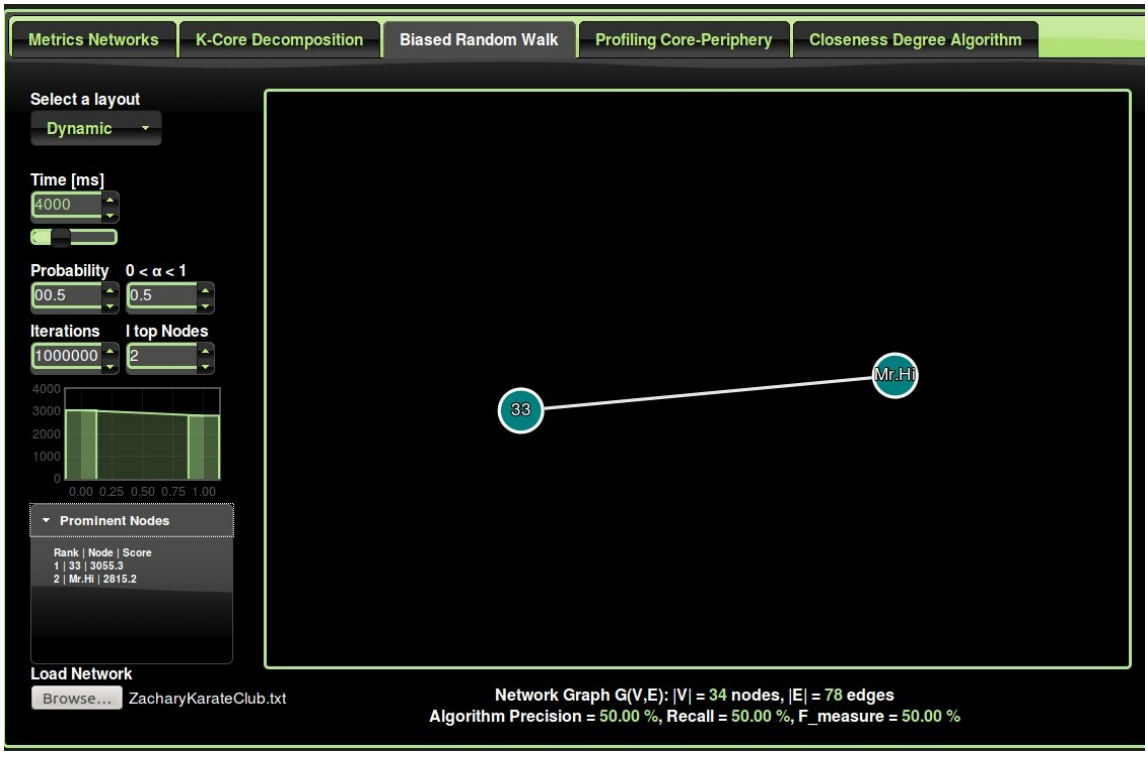


Figure 10: Biased Random Walk algorithm applied to the Zachary Karate club network.

All the previous experiments have been summarized in the following Table 1.

Zachary Karate Club Network, 34 nodes, 78 edges, k = 2						
K-Core Decomposition 100% of the prominent nodes are contained in the highest Core 4						
	Metrics, Indicators	**Biased Random Walk**	**Random Walk**	**Profiling Core-Periphery**	**Degree Algorithm**	**Closeness Degree**
L=10	Precision P, L=10	33.33%	33.33%	33.33%	33.33%	33.33%
	Recall R	100.00%	100.00%	100.00%	100.00%	100.00%
	Fmeasure F	33.33%	33.33%	33.33%	33.33%	33.33%
L=5	Precision P, l=5	57.14%	57.14%	57.14%	57.14%	57.14%
	Recall R	100.00%	100.00%	100.00%	100.00%	100.00%
	Fmeasure F	57.14%	57.14%	57.14%	57.14%	57.14%
L=3	Precision P, L=3	40.00%	40.00%	80.00%	80.00%	80.00%
	Recall R	50.00%	50.00%	100.00%	100.00%	100.00%
	Fmeasure F	40.00%	40.00%	80.00%	80.00%	80.00%
L=2	Precision P, L=2	50.00%	50.00%	100.00%	100.00%	50.00%
	Recall R	50.00%	50.00%	100.00%	100.00%	50.00%
	Fmeasure F	50.00%	50.00%	100.00%	100.00%	50.00%

Table 1: Summary of all the algorithms' outcome carried out on the Zachary Karate club.

Notice that the random walk algorithm is obtained by simply setting the p=0.5 and $\alpha = 1$, where we neglect the effect of neighborhood density effect and retain only the degree of centrality effect.

A visual histogram chart representation in the Figure 11 summarizes all the algorithms' performance.

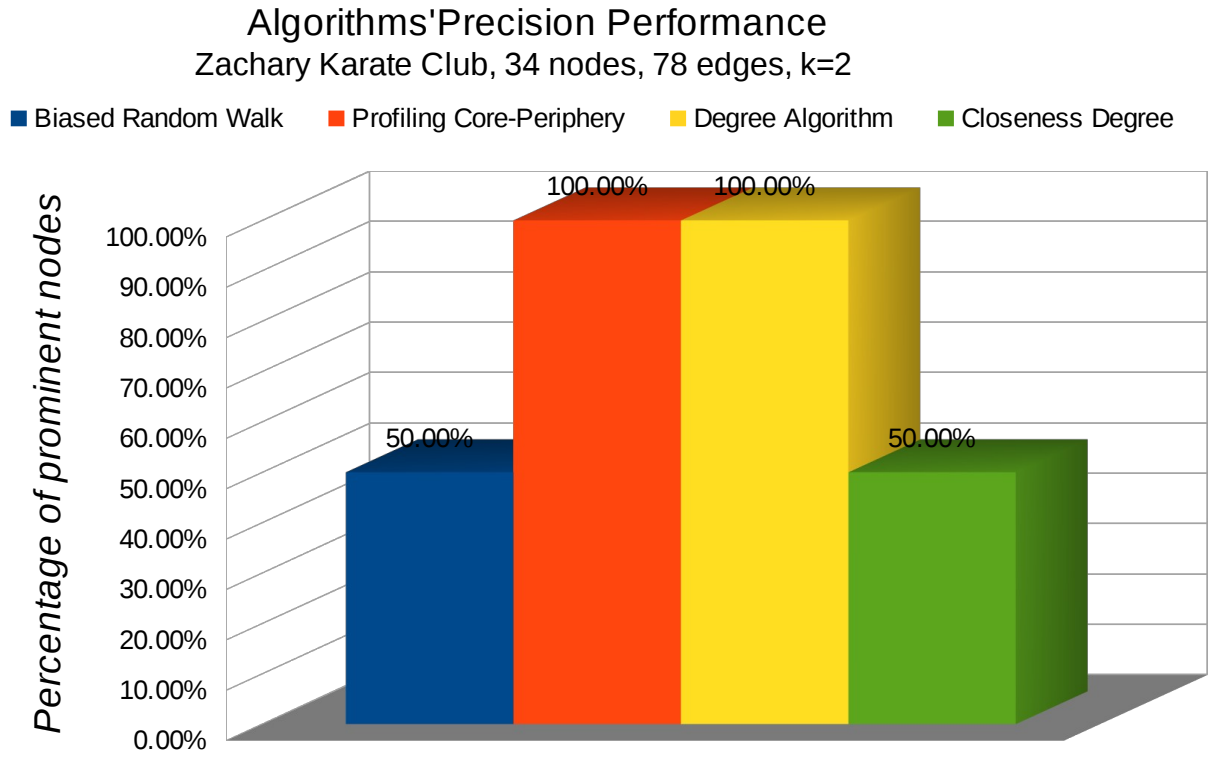


Figure 11: Histogram Chart Algorithms' Performance (Zachary Karate Club Network)

From the algorithm results applied to the Zachary Karate Club, we found that the Profiling Core-Periphery algorithm outperforms the other methods and retrieves 100 % of the prominent nodes (l=k=2).
However the maximum reached par the random walk is 80% of the prominent nodes with l=k=3.

5.2 Second dataset: Thurman office

Thurman spent 16 months observing the interactions among employees in the overseas office of a large international corporation. During this time, two major disputes erupted in a subgroup of fifteen people. Thurman analyzed the outcome of these disputes in terms of the network of formal and informal associations among those involved. This social network of the 15 office workers reported by Thurman (1979) is used to evaluate the algorithms. The network is shown in Figure 12.

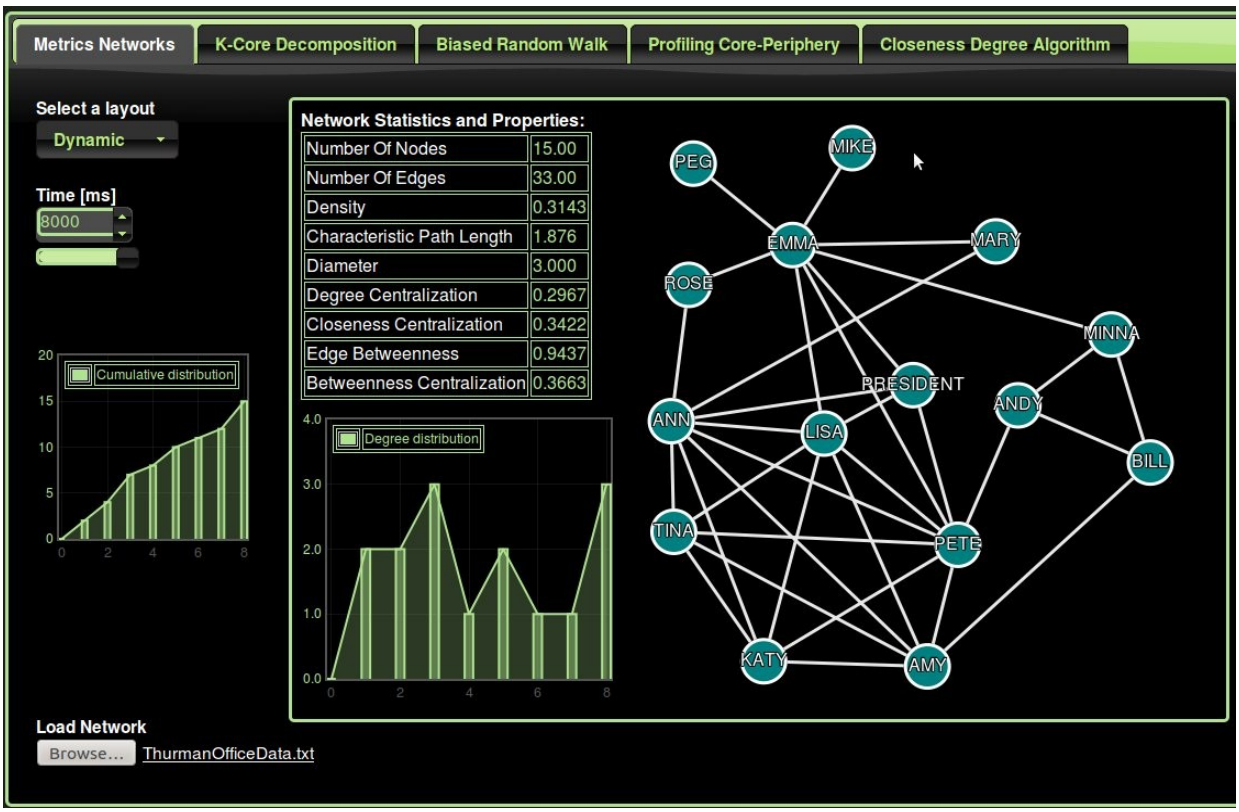


Figure 12: Network visualization and Statistics of the Thurman office.

According to Thurman (1979), Pete and Emma are characterized as the center of their social circle. Therefore, Pete and Emma will be our prominent nodes to evaluate different algorithms. By applying different algorithms to this social network with l=13, 10, 5, 3 top nodes, we obtain the following results summarized in Table 2

Thurman Office Network, 15 nodes, 33 edges, k = 2						
K-Core Decomposition 50% of the prominent nodes are contained in the highest Core 5 and 50% Core 3.						
	Metrics, Indicators	Biased Random Walk	Random Walk	Profiling Core-Periphery	Degree Algorithm	Closeness Degree
L=13	Precision P, L=13	26.67%	26.67%	26.67%	26.67%	26.67%
	Recall R	100.00%	100.00%	100.00%	100.00%	100.00%
	Fmeasure F	26.67%	26.67%	26.67%	26.67%	26.67%
L=10	Precision P, L=10	16.67%	16.67%	33.33%	33.33%	33.33%
	Recall R	50.00%	50.00%	100.00%	100.00%	100.00%
	Fmeasure F	16.67%	16.67%	33.33%	33.33%	33.33%
L=5	Precision P, L=5	28.57%	28.57%	57.14%	57.14%	57.14%
	Recall R	50%%	50%%	100.00%	100.00%	100.00%
	Fmeasure F	28.57%	28.57%	57.14%	57.14%	57.14%
L=3	Precision P	40.00%	40.00%	80.00%	80.00%	80.00%
	Recall R	50%%	50%%	100%%	100%%	100%%
	Fmeasure F	40%%	40%%	80%%	80%%	80%%
L=2	Precision P	0.00%	50.00%	50.00%	50.00%	100.%%
	Recall R	0.00%	50.00%	50.00%	50.00%	100.00%
	Fmeasure F	0.00%	50.00%	50.00%	50.00%	100.00%

Table 2: Summary of all the algorithms' outcomes carried out on the Thurman Office.

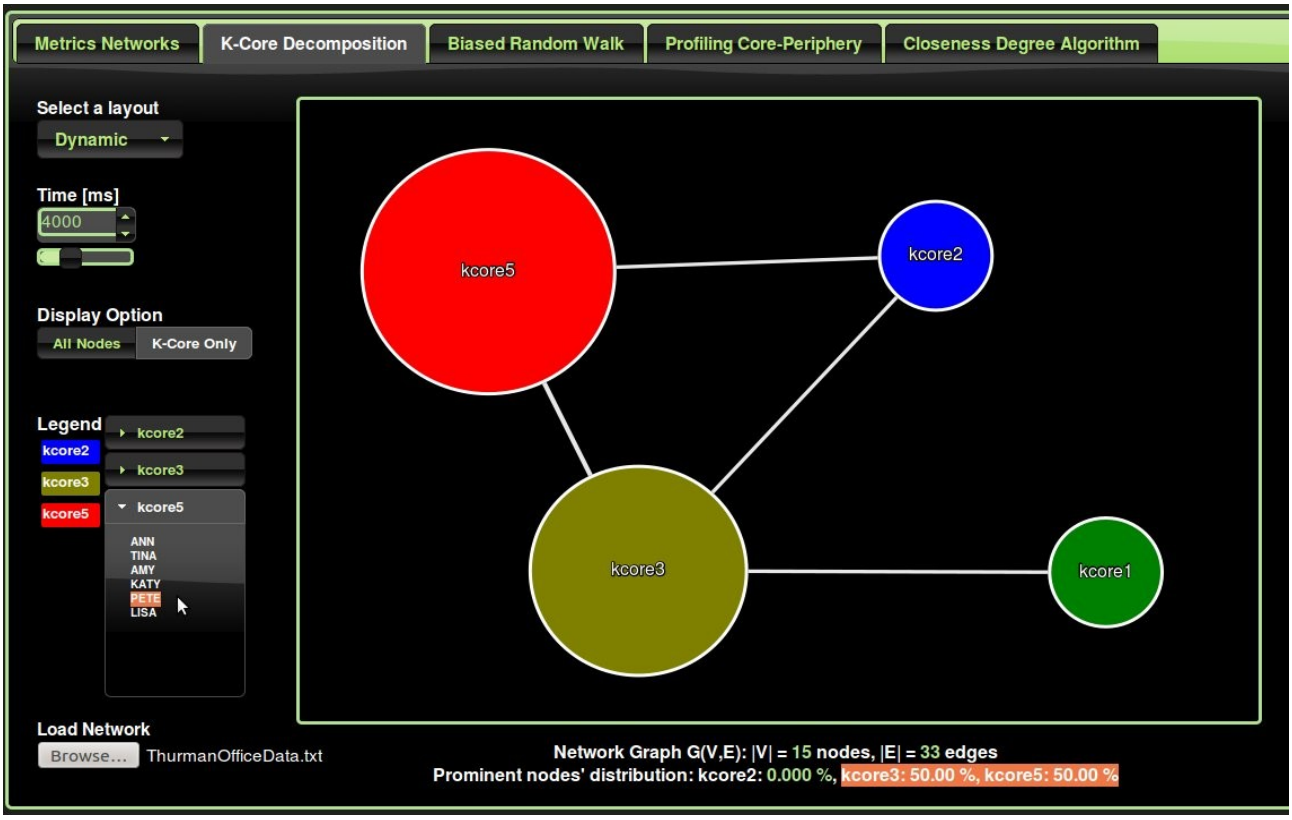


Figure 13: K-Core algorithm applied to the Thurman Office network (Core representation)

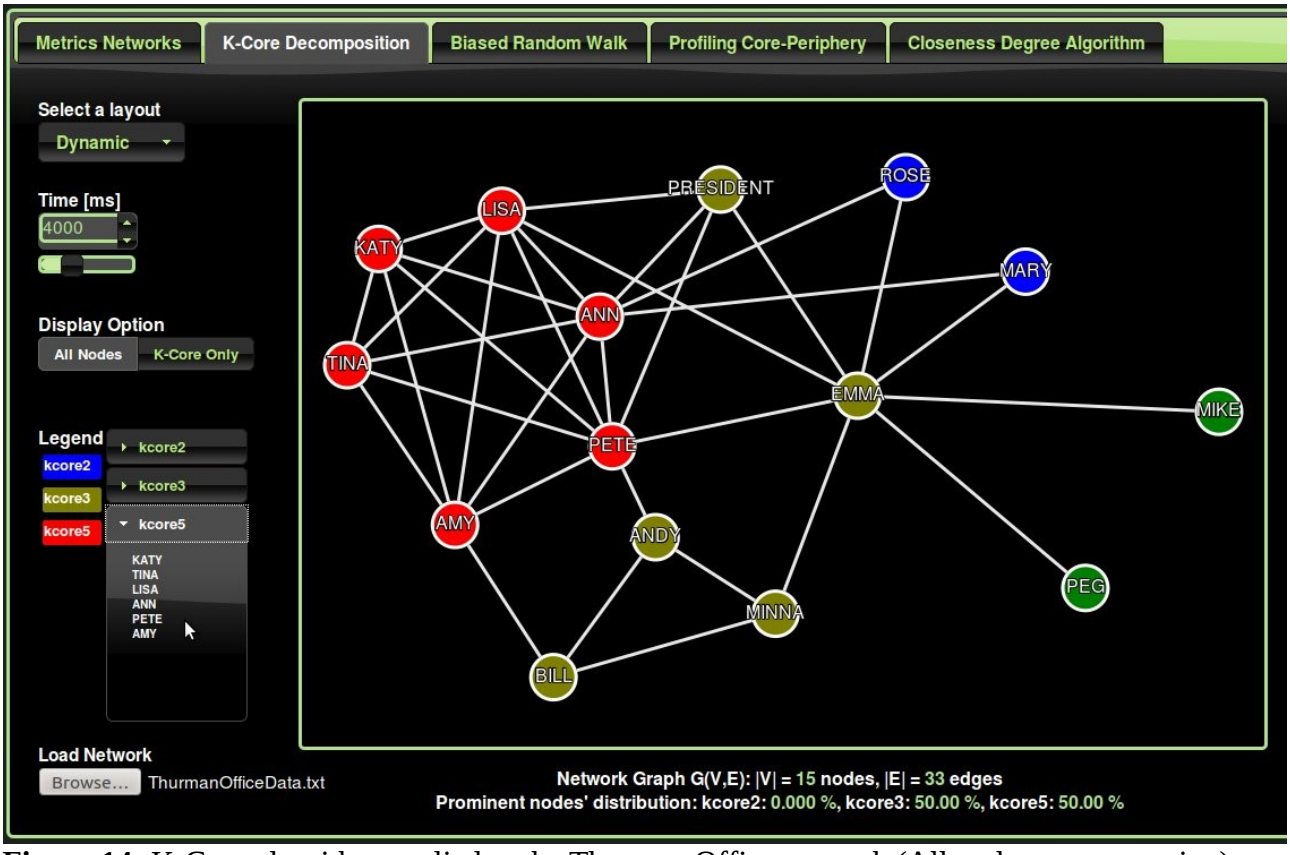


Figure 14: K-Core algorithm applied to the Thurman Office network (All nodes representation)

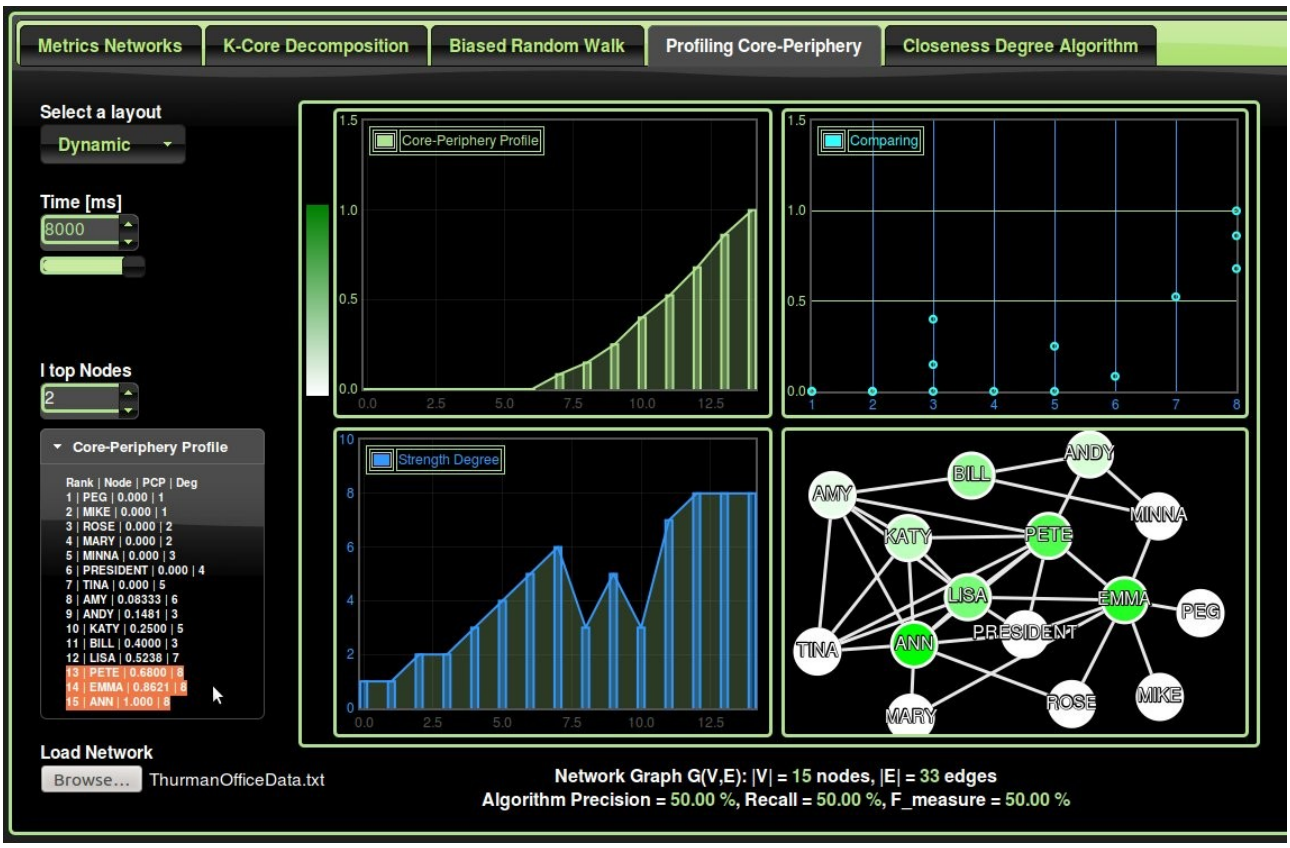


Figure 15: Profiling Core-Periphery algorithm applied to the Thurman Office network.

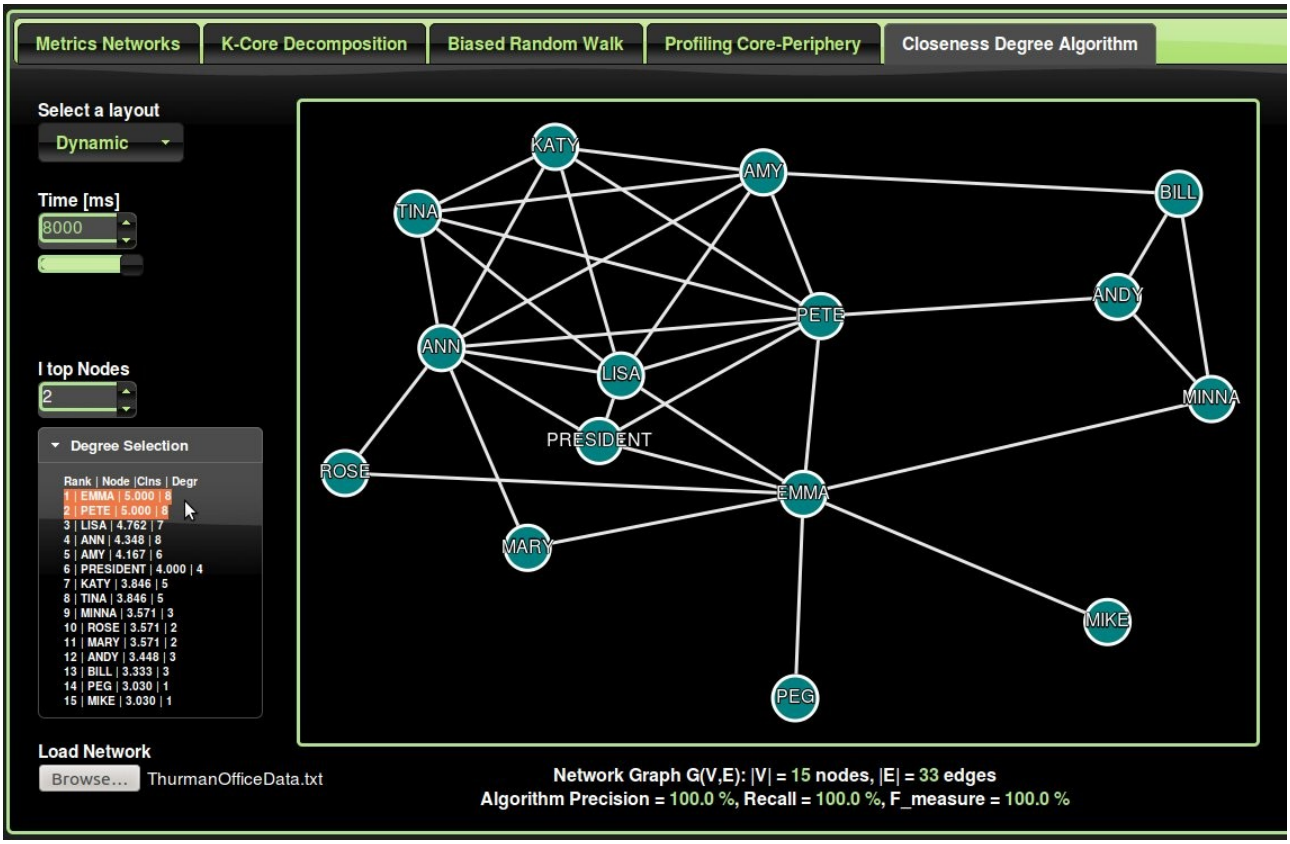


Figure 16: Biased Random Walk algorithm applied to the Thurman Office network.

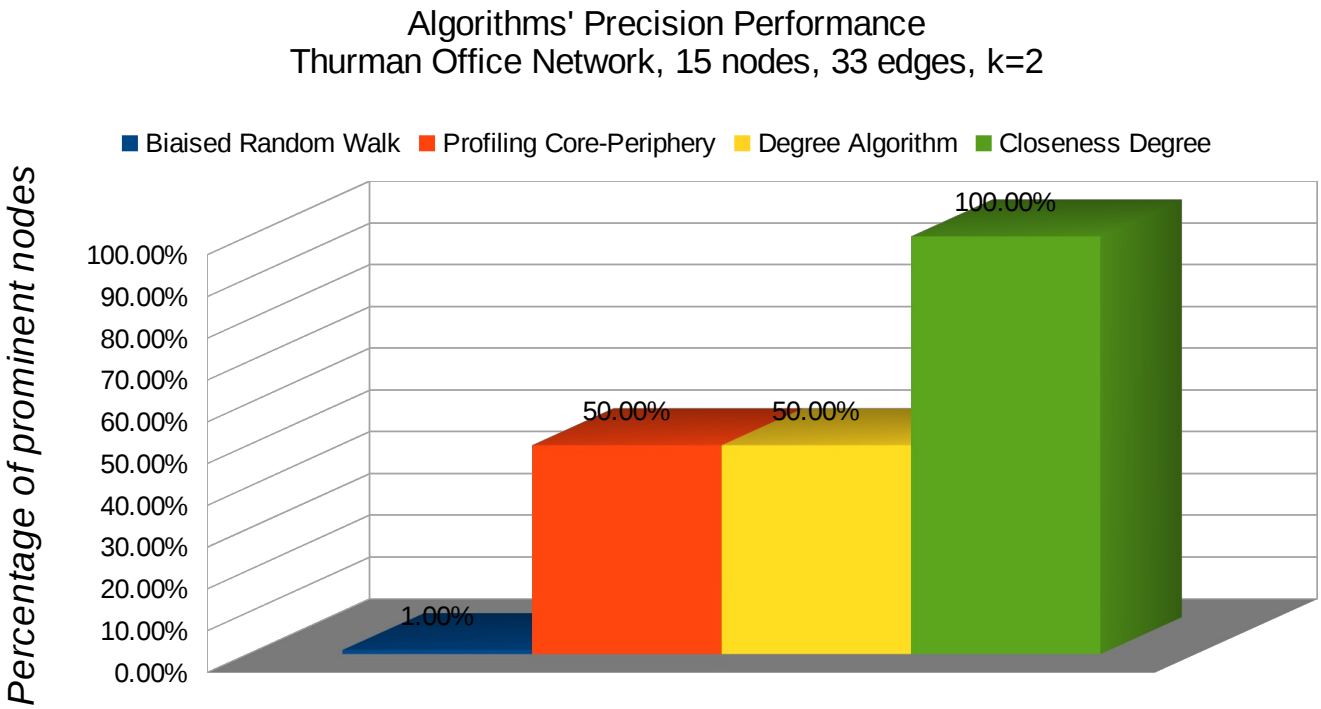


Figure 17: Histogram Chart Algorithms' Performance l=k=2 (Thurman Office Network)

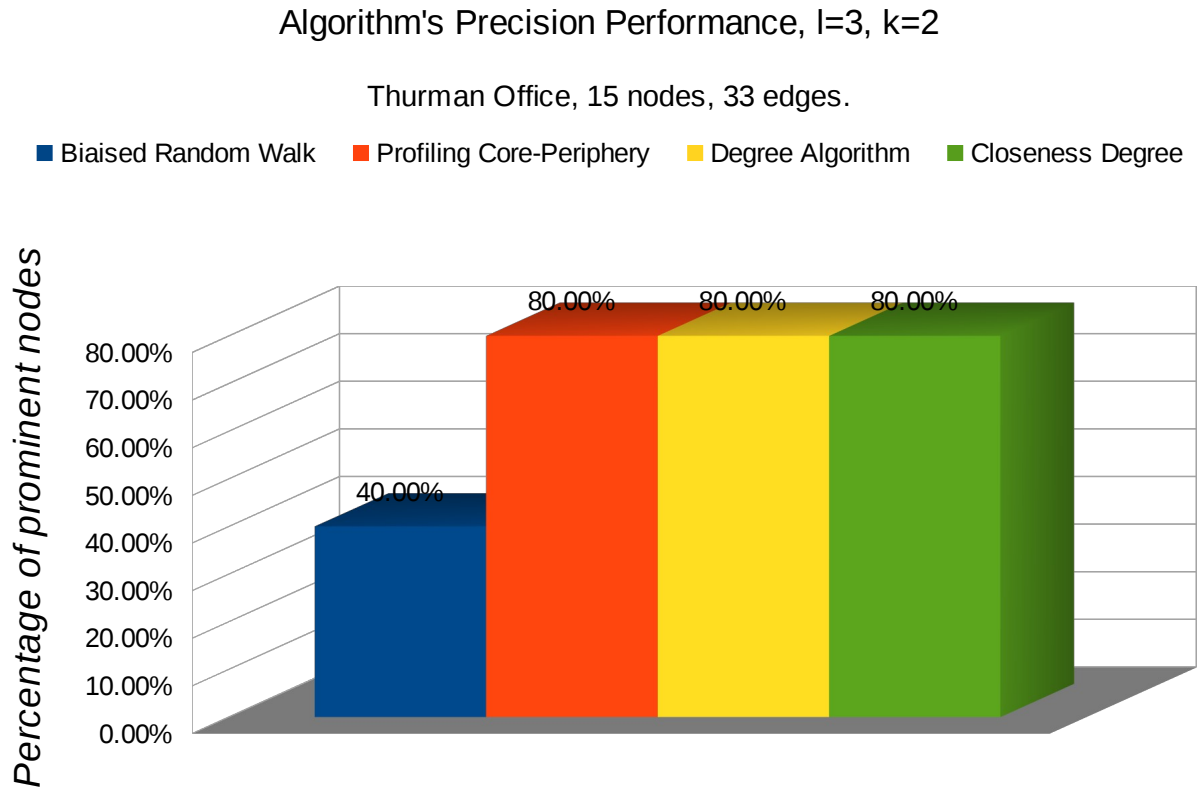


Figure 18: Histogram Chart Algorithms' Performance l=3, k=2 (Thurman Office Network)

Here again, the Profiling Core-Periphery outperforms by far the other methods. The PCP method reaches 80% of the prominent nodes, where the Biased Random Walks method struggles to barely reach 40%.

5.3 Third dataset: Social network of bottlenose dolphin

Social network of bottlenose dolphin living in Doubtful Sound, New Zealand.
This dolphin dataset was collected by DL, Olivier J. Boisseau for the long-term research program of the University of Otago-Marine Mamma Research Group. In their research they have identified communities and subcommunities within the dolphin population. More interesting, they have also identified brokers who act as links between subcommunities and who appear to be crucial to the social cohesion of the population as a whole. These brokers with highest betweenness fall, not surprisingly, on the boundary between the communities in the network and will represent the prominent nodes that our algorithms should identify.

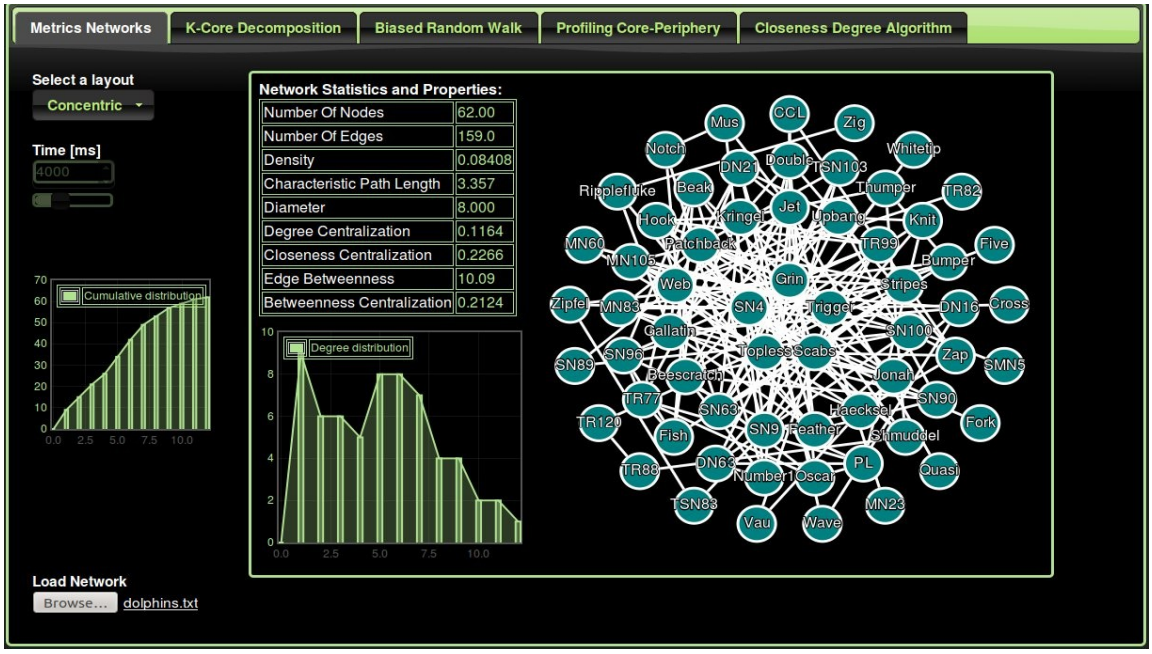


Figure 19: Network visualization and Statistics of the bottlenose dolphins network.

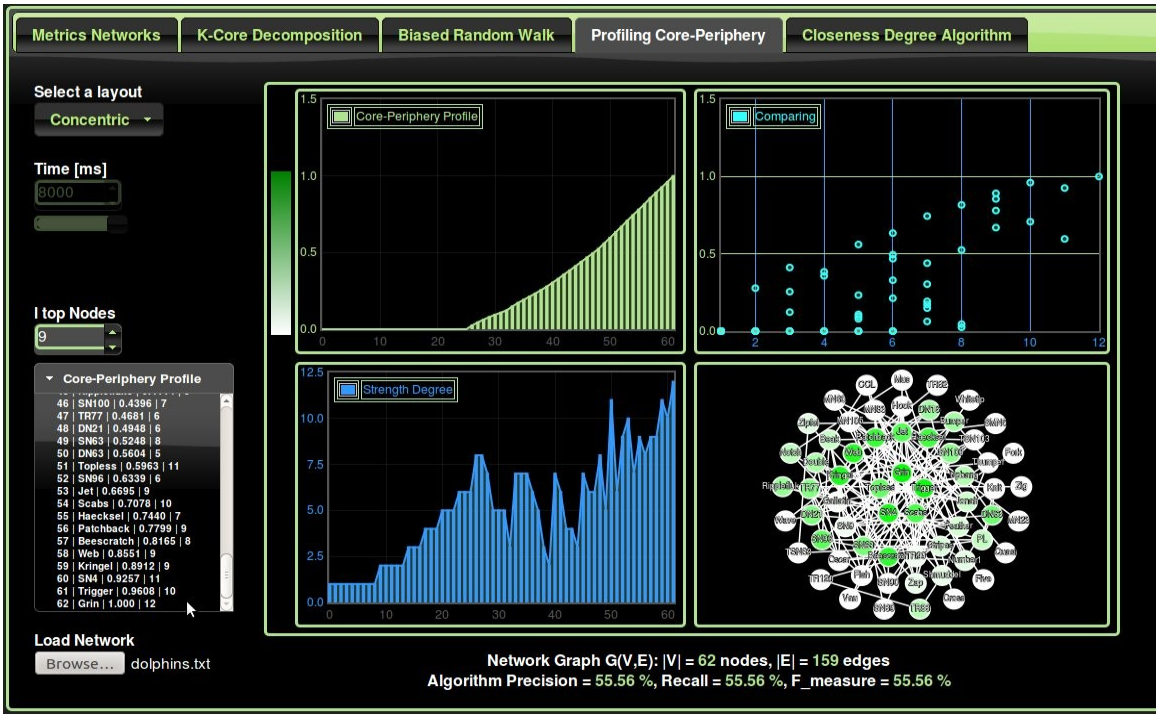


Figure 20: Profiling Core-Periphery algorithm applied to the bottlenose dolphins network.

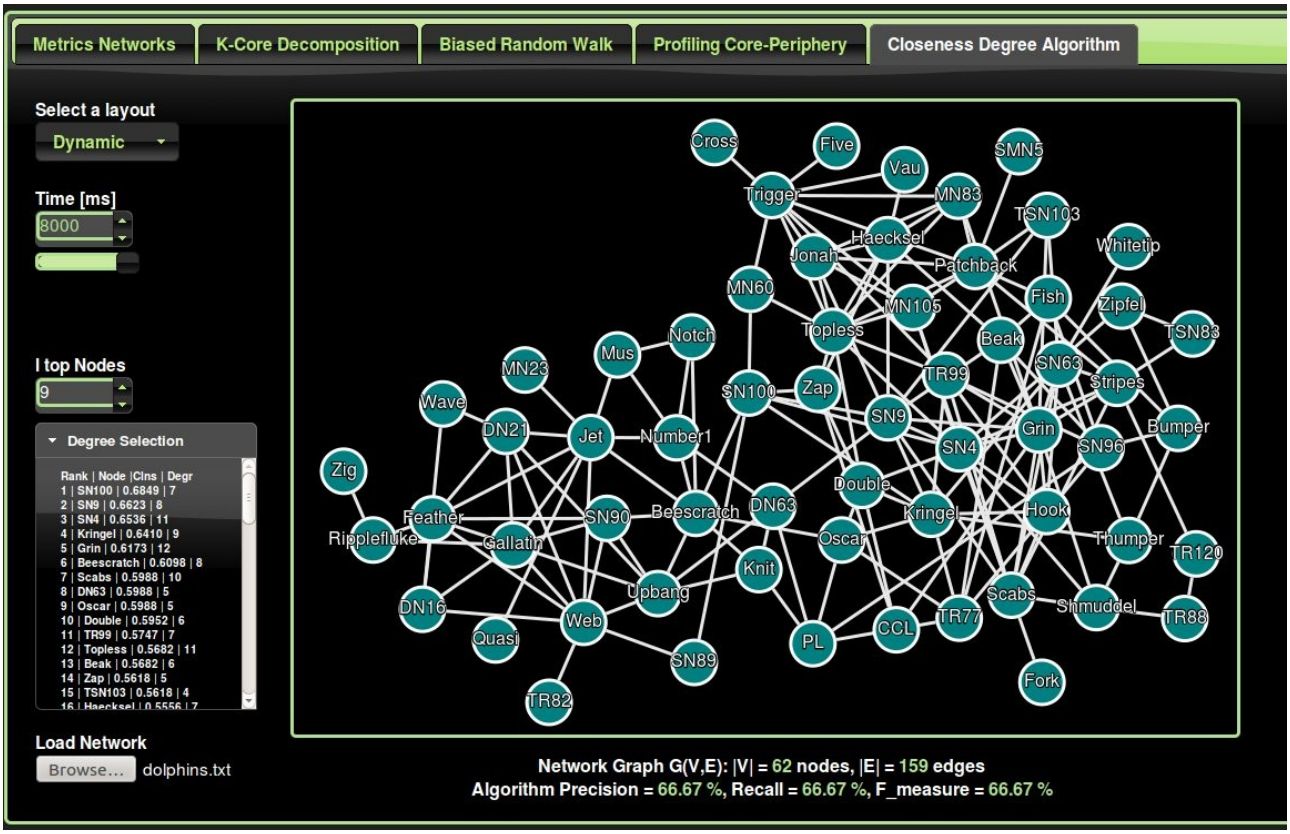


Figure 21: Biased Random Walk algorithm applied to the bottlenose dolphins network.

Using this dataset to evaluate different algorithms, I obtain the following results: Table 3

Dophins network, 62 nodes, 159 edges, k = 9						
K-Core Decomposition 55.56% of the prominent nodes are contained in the highest Core 4.						
	Metrics, Indicators	**Biased Random Walk**	**Random Walk**	**Profiling Core-Periphery**	**Degree Algorithm**	**Closeness Degree**
L=25	Precision P	29.41%	29.41%	47.06%	47.06%	41.18%
	Recall R	55.56%	55.56%	88.89%	88.89%	77.78%
	Fmeasure F	29.41%	29.41%	47.06%	47.06%	41.18%
L=20	Precision P, l=20	20.69%	27.59%	48.28%	55.17%	41.38%
	Recall R	33.33%	44.44%	77.78%	88.89%	66.67%
	Fmeasure F	20.69%	27.59%	48.28%	55.17%	41.38%
L=15	Precision P	25.00%	25.00%	50.00%	58.33%	50.00%
	Recall R	33.33%	33.33%	66.67%	77.78%	66.67%
	Fmeasure F	25.00%	25.00%	50.00%	58.33%	50.00%
L=9	Precision P	11.11%	11.11%	55.56%	44.44%	66.67%
	Recall R	11.11%	11.11%	55.56%	44.44%	66.67%
	Fmeasure F	11.11%	11.11%	55.56%	44.44%	66.67%
L=5	Precision P	14.29%	14.29%	57.14%	28.57%	57.14%
	Recall R	11.11%	11.11%	44.44%	22.22%	44.44%
	Fmeasure F	14.29%	14.29%	57.14%	28.57%	57.14%
L=3	Precision P	16.67%	16.67%	33.33%	16.67%	50.00%
	Recall R	11.11%	11.11%	22.22%	11.11%	33.33%
	Fmeasure F	16.67%	16.67%	33.33%	16.67%	50.00%

Table 3: Summary of all the algorithms' results carried out on the bottlenose dolphins network.

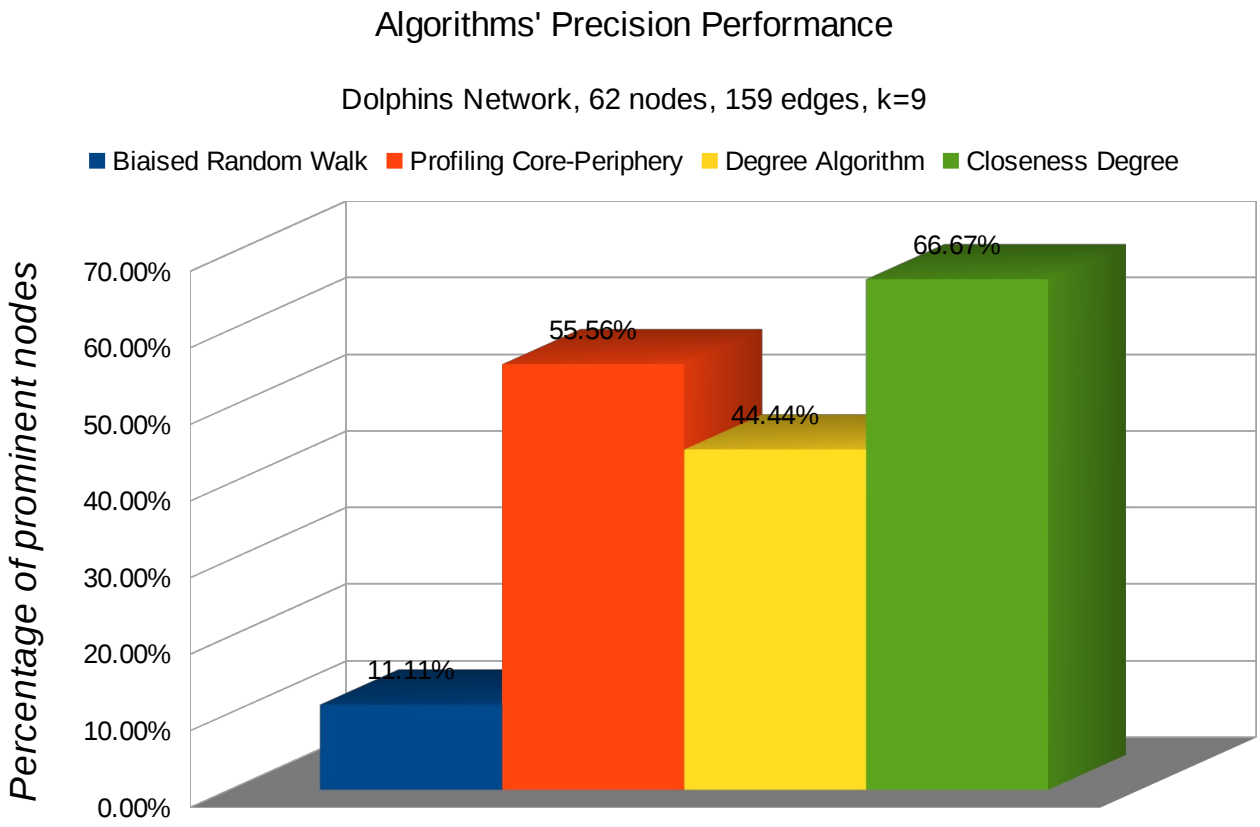


Figure 22: Algorithms' Performance l=k=9 (bottlenose dolphins network)

Here again, the Profiling Core-Periphery outperforms by far the other methods. The PCP method reaches almost 60% of the prominent nodes, where as the Biased Random Walks method struggles to hardly reach 12%.

5.4 Fourth dataset: Books from Amazon 2004.

Books about US politics: A network of books about US politics published around the time of the 2004 presidential election and sold by the online bookseller Amazon.com. Edges between books represent frequent copurchasing of books by the same buyers. The network was compiled by V. Krebs and is unpublished, but can found on Krebs' web site. http://www.orgnet.com/

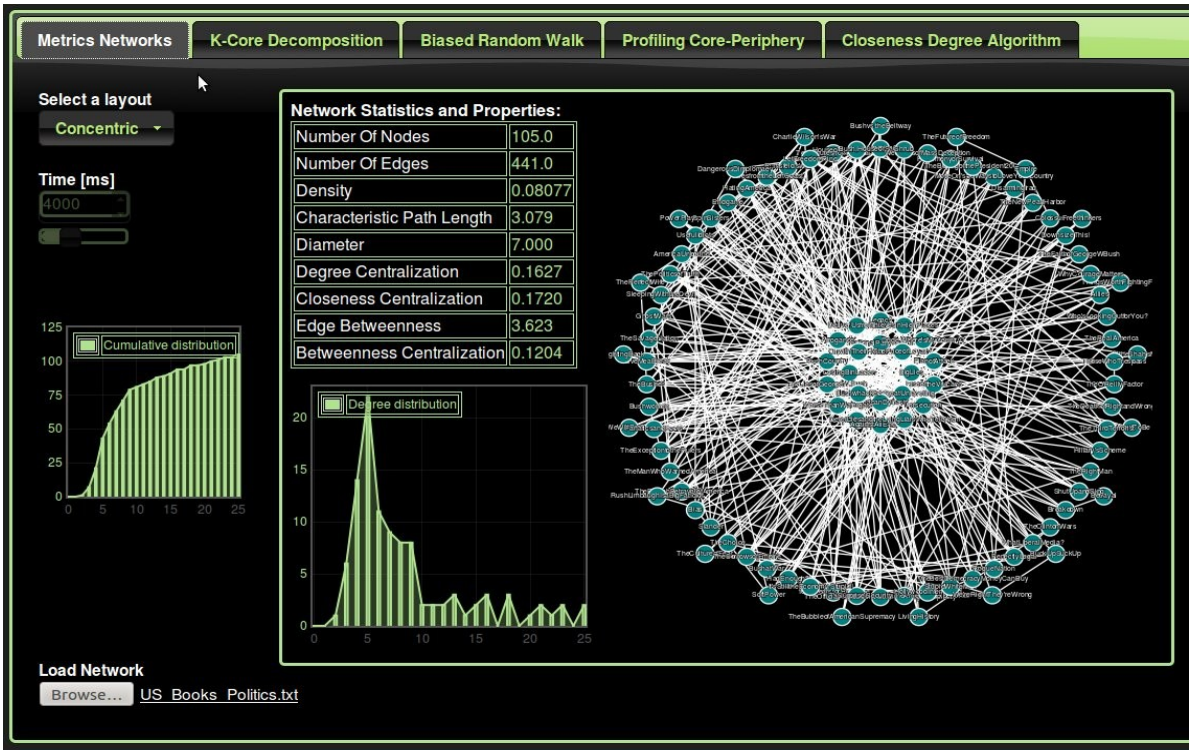


Figure 23: Network visualization and Statistics of the US books network.

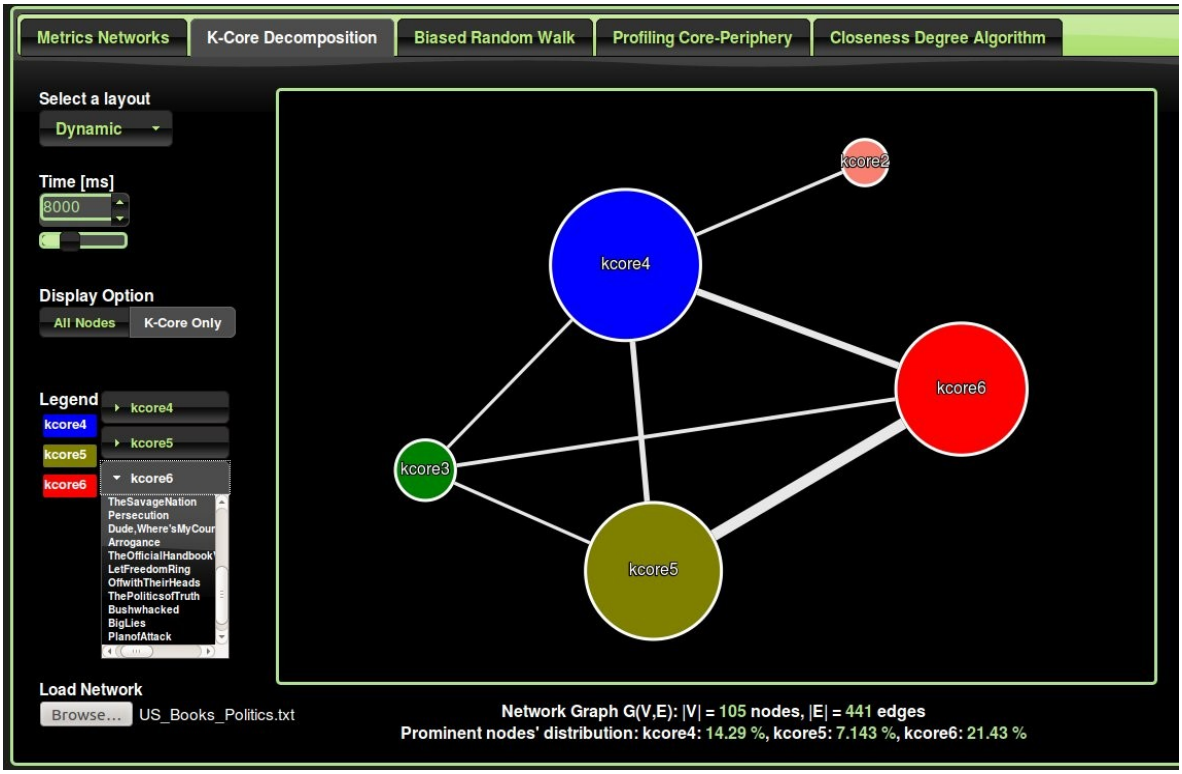


Figure 24: K-Core algorithm applied to the US books network (Core representation)

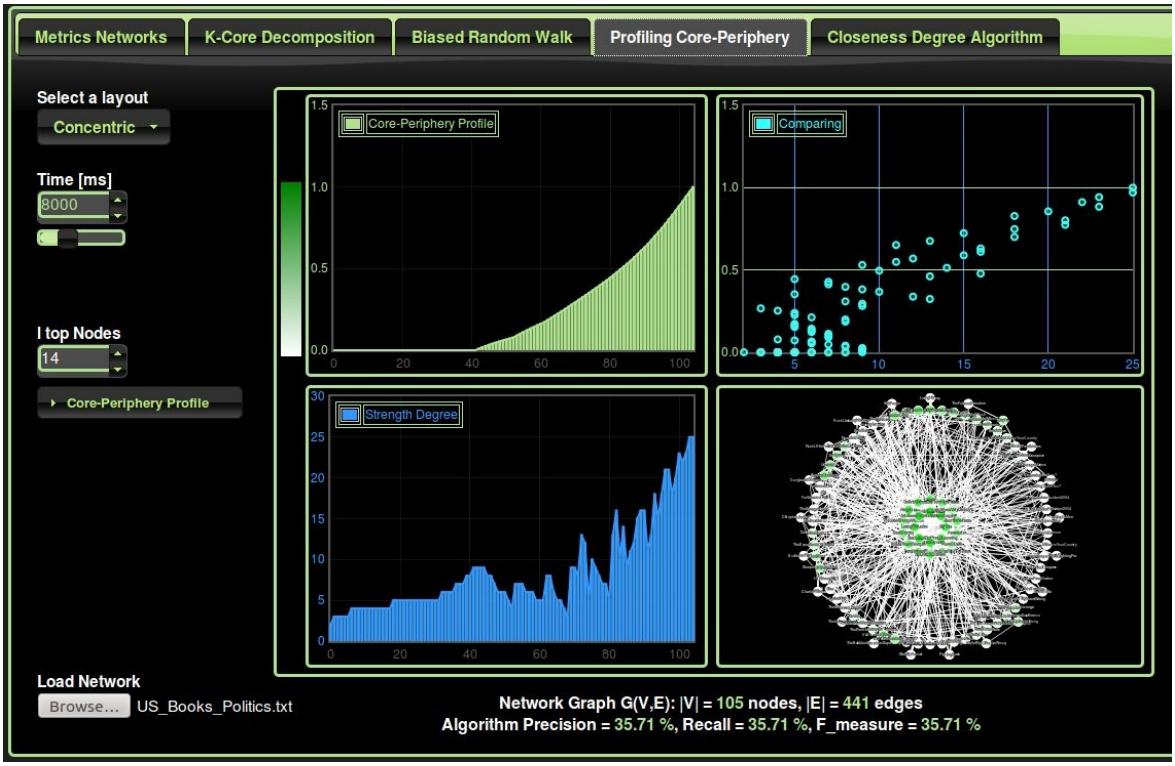


Figure 25: Profiling Core-Periphery algorithm applied to the US books network.

US Books Network, 105 nodes, 441 edges, k = 14					
K-Core Decomposition 21.43% of the prominent nodes are contained in the highest Core 6, 7.143% Core 5, and 14.29% in kcore 4.					
	Metrics, Indicators	**Biased Random Walk**	**Random Walk**	**Profiling Core-Periphery**	**Closeness Algorithm**
L=14	Precision P, L=14	21.43%	21.43%	35.71%	28.57%
	Recall R	21.43%	21.43%	35.71%	28.57%
	Fmeasure F	21.43%	21.43%	35.71%	28.57%

Table 4: Summary of all the algorithms' outcomes carried out on the US books network.

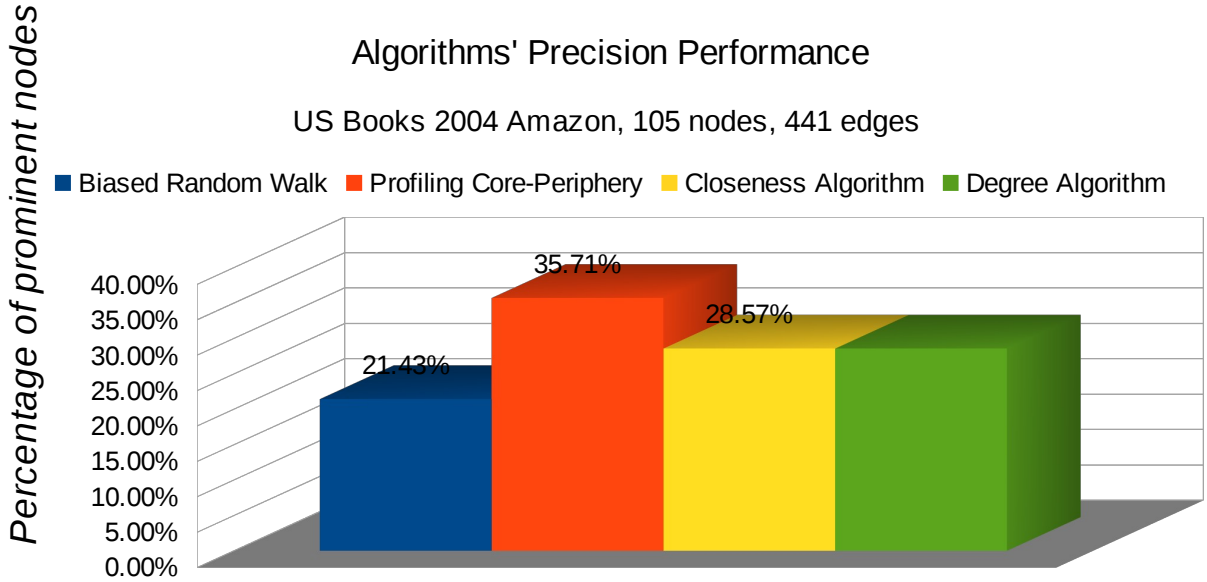


Figure 26: Histogram Chart Algorithms' Performance l=k=14 (US books network)

To obtain the prominent books, I had to first gather the Amazon Best Sellers Rank for each Book, and then sort the whole list of books based on the rank criterion to take only the top 14.

5.5 Fifth dataset: Jazz musicians bands network

The data was obtained from the Red Hot Archive digital database. The network dataset include 198 bands that performed between 1912 and 1940, with most of the band in 1920's.
We consider the collaboration between bands, where two bands are connected if they have a musician in common. We do not know the prominent nodes in this network, however , if we sort the bands by their degrees, in the descending order, we notice that the first four nodes are the outlying nodes with noticeably higher degrees. These nodes could represent bands that had great notoriety among the jazz community, bands that existed during the entire 28 years period, or possibly bands that had high turnover of musicians.

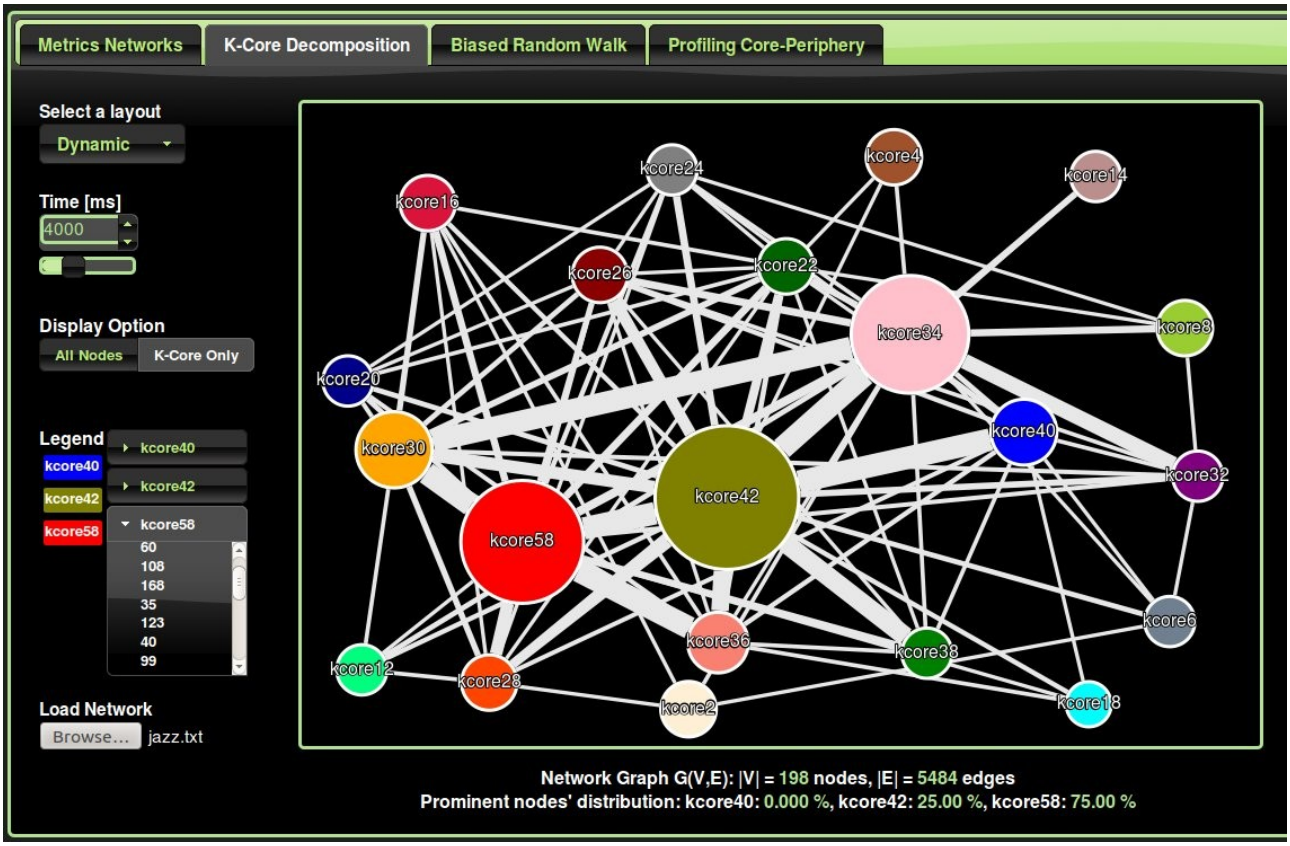


Figure 27: K-Core algorithm applied to the Jazz band musicians network (Core representation)

Not surprising at all, if we apply the k-core decomposition to this Jazz network, these outlying Jazz bands belong to the highest cores. As shown in Figure 27, three quarter (75%) of these highly connected bands belong to the core 58 and one quarter(25%) to the core 42.

Chapter 6: Conclusion

In this project, we have developed an online web application, a service to users interested in visualizing, analyzing, and detecting prominent nodes in their networks. The user can connect to the server using any browser, gather statistics and visualize their network in different layouts. The network uploaded into the server is a simple text file, using an intuitive and simplified edges list graph notation.

From the four data sets tested, in detecting most prominent nodes, we found that the complex algorithms do not add a significant value to the simple closeness and degree selection algorithm.

In general the Profiling Core-Periphery algorithm outperforms the Biased Random Walk and we obtain almost the same performance between the Profiling Core-Periphery and a simple closeness/degree algorithm. However, we have observed that the closeness centralization measure is more costly in time due to the fact that we need to calculate the sum of the geodesic distances from ego to all others in the network. The complexity of calculating centrality measures such as the closeness centrality is in the order of O(mn) or worse [10] ($O(n^3)$) where m is the number of edges and n the number of nodes. Therefore the time complexity directly depends on the density of the network, in other words, how sparse is the network. On the other hand, the time complexity of a simple degree selection algorithm is of the n order O(n) but does not take into account the neighborhood effect and could select a peripheral node with high degree.

In the bias random walk algorithm, it is worth repeating that in our experiments, we have followed the article's recommendation [1] for the parameter values. However, it is very likely that these parameters need to be adapted to the context and environment.

To sum up, in all the experiments carried out, the biased random walks and profiling core periphery algorithm were faster due to my design and implementation ($O(n^2)$ worst case) than the baseline algorithm (closeness/degree criterion $O(n^3)$). Therefore the Profiling Core-Periphery offer a good compromise between performance/precision and time execution/search of the prominent nodes.

This study is only a start, and should not be generalized as only four data sets have been evaluated. Plus the statistical significance of these data sets is discutable. Also, the algorithms are actually restricted to the unweighted and undirected graphs. Therefore, we should first extend the algorithms to allow numerous and various kind of networks to be tested and evaluated, in order to obtain more accurate and significant statistical results.

Appendix 0: Contents of CD

README.txt
Presentation PowerPoints Slides: MSc_Project_Presentation_08_2014.odp
Articles/References: Article_References/
PDF Final Report: MSc_Project_Report_amtir_09_2014.pdf
DOC Format Final Report: MSc_Project_Report_amtir_09_2014.doc
WAR Web Application ARchive: MSc_Project_JavaEE_amtir_09_2014.war
JAVA Code Documentation: JavaDoc/
Networks Samples: Data_Samples_Networks/
Eternal libraries: External_Libraries/
Unit/Suite/Cases/Test: Unit_Cases_Suite_Tests/

Versioning Control System
The code has been regularly save under Subversion, a versioning control system, that is very similar and compatible to Concurrent Versions System (CVS). As a developer I used SVN Subversion to maintain current and historical version of files such as source code, web pages, and documentation.

The school provides a subversion service that acts as a server SVN, that is available to staff and students. A copy of my work could be found under the following link:

https://codex.cs.bham.ac.uk/svn/axm1064/MScProject2014/

NLOC Number Lines Of Code

LANGUAGE	**NLOC**
Java-Script	1500
Java	2000
HTML, JSP	500
CSS	600

Reasons for choosing Java language
Extensive standardized APIs (Application Programming Interfaces)
Better portability than other languages across operating systems
Supports web based applications (Servlet, JSP), distributed applications (sockets, RMI) and network protocols (HTTP)
Object Oriented programming OOP
Built-in support for multi-threading, socket communication, and memory management (automatic garbage collection).

Appendix 1: Test Cases, JUnits

Test	Description/Scenario	Expected result	Actual Result	Pass/Fail
Empty set of nodes	Handle the case where the user enter an empty file (empty set of nodes)	The front side (browser/JavaScript) should handle this case and silently bypassing, without even bothering the server.	As expected	Pass Application stable
One simple node	Handle the case where the user enter a unique single node. Boundary limit cases testing	A unique node JSON object request/response is exchanged between the client and the server and this for all the algorithms. Stable application.	As expected	Pass
Two nodes with one edge	Handle the case where the user enter Two nodes linked by an edge. Boundary limit cases testing	Application stable; the results returned are correct and accurate for all the algorithms.	As expected	Pass
Two independent nodes	Handle the case where the user enter Two independent nodes. Boundary limit cases testing	Application stable; the results returned are correct and accurate for all the algorithms.	As expected	Pass
Three nodes with two edges	Handle the case where the user enter three nodes linked by two edges. Boundary limit cases testing	Application stable; results correct and accurate for all the algorithms.	As expected	Pass
Three independent nodes without any edge	Handle the case where the user enter three independent nodes. Boundary limit cases testing	Application stable; the results returned are correct and accurate for all the algorithms.	As expected	Pass
Multiple redundant nodes,	Handle the case of redundant data,	Application stable; and correct		

single, linked with other nodes	nodes, edges.	and accurate for all the algorithms.	As expected	Pass
Circle network	Typical network graph to test evaluate the application.	Application stable; and correct and accurate for all the algorithms.	As expected	Pass
Star Network	Typical network graph to test evaluate the application.	Application stable; and correct and accurate for all the algorithms.	As expected	Pass
Line Network	Typical network graph to test evaluate the application.	Application stable; and correct and accurate for all the algorithms.	As expected	Pass
Stress test Max load	Quantify the maximum data exchangeable between the client and the server.	While the application is stable; and the results are correct and accurate for all the algorithms.	As expected	Pass
Comments Single and multiple comment line	Ensure that the user can comment different section of its network file.	Application stable.	As expected Application stable	Pass
Tabs and Space characters	Handle users' misbehavior in translating and writing their network	Application stable; and correct and accurate for all the algorithms.	As expected	Pass
Prominent nodes test	Ensure that the user can enter the prominent nodes in the network file, and collect statistics about the algorithms' precision and accuracy.	Application stable; Statistics are correct and accurate for all the algorithms.	As expected	Pass

All the previous unit test cases are grouped in a JUnit test suit. Each time a new functionality or feature is developed, its unit test case is added to the test suit, which is executed again, to test the stability, correctness and accuracy of the whole application.

Test cases, regression JUnit

Correctness: *To ensure the algorithm return the correct result, deliberate input networks were used with the web-application, and the outcome was compared with the expected result.*

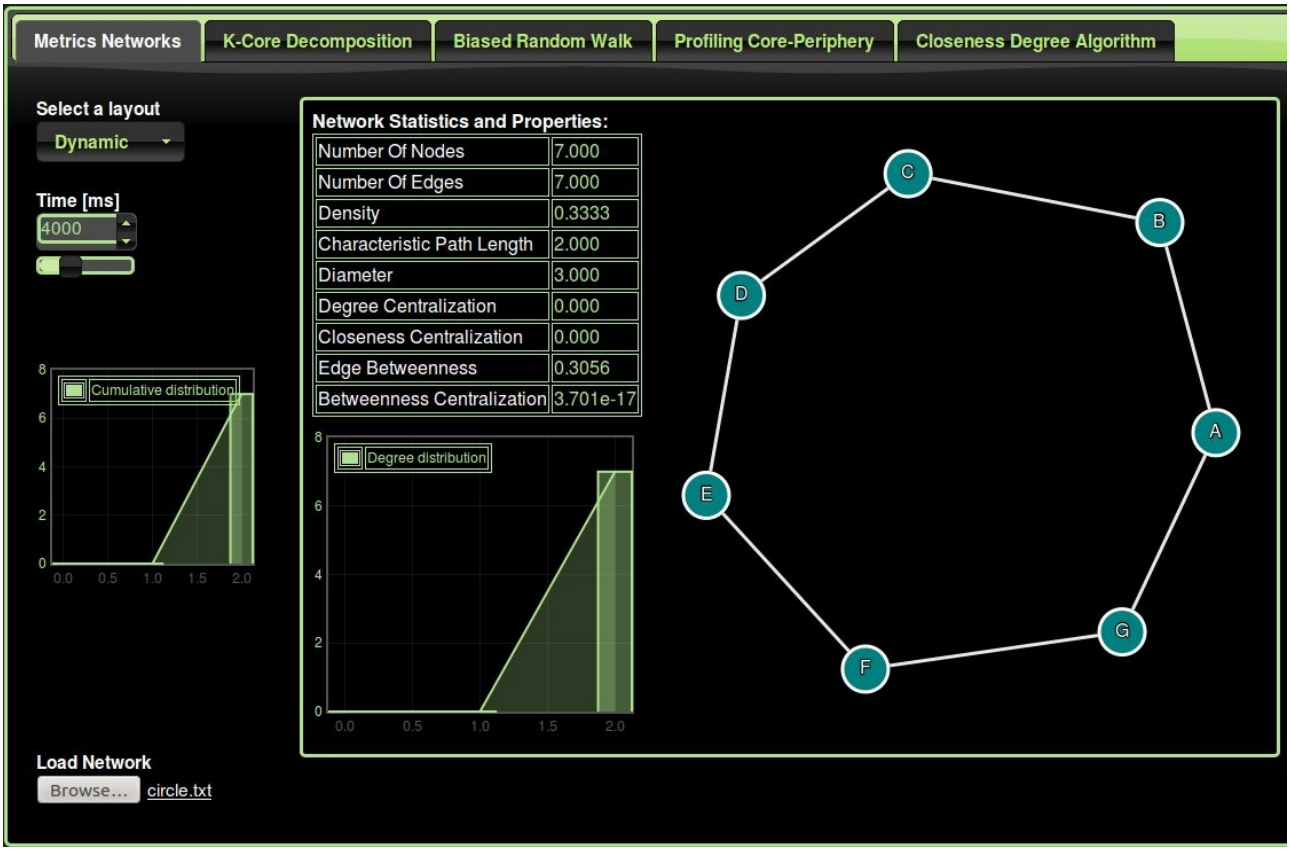


Figure 28: Circle: 0% on all measures (degree and closeness) of centralization.

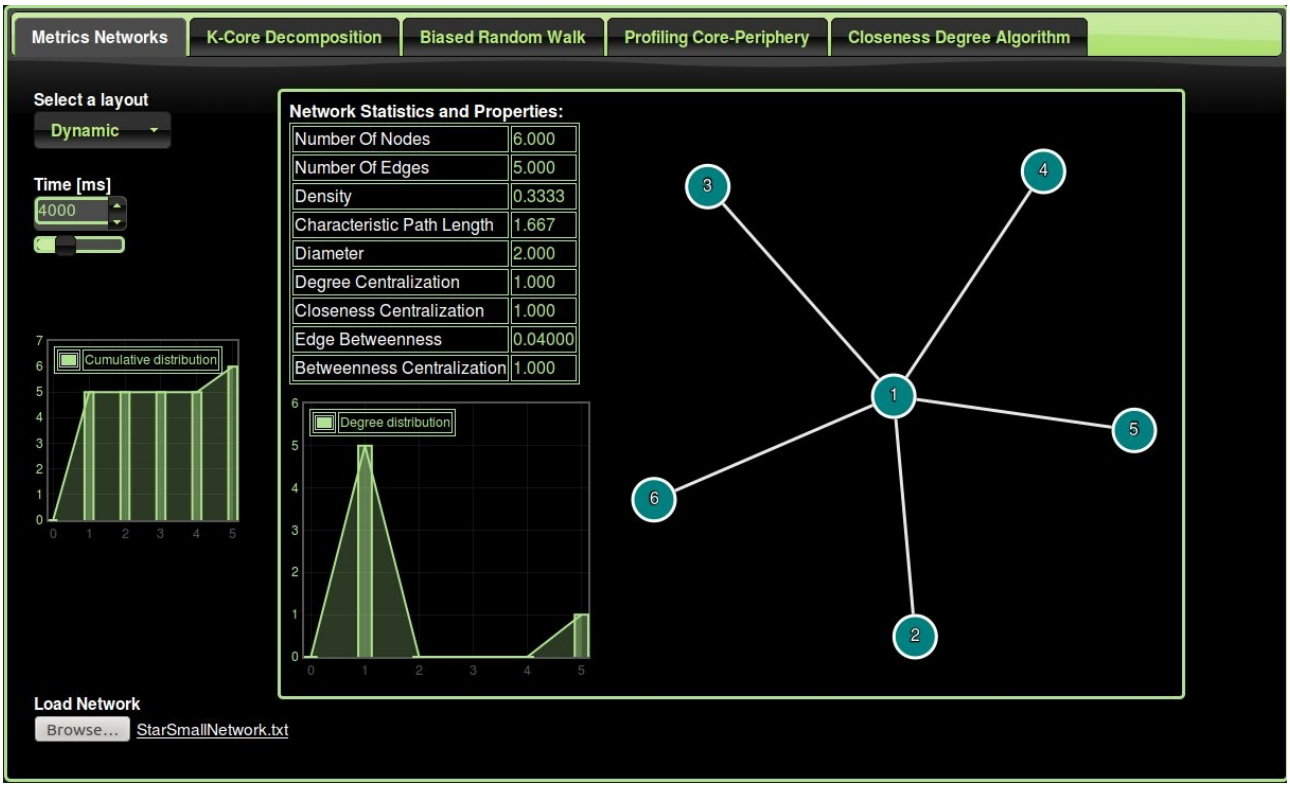
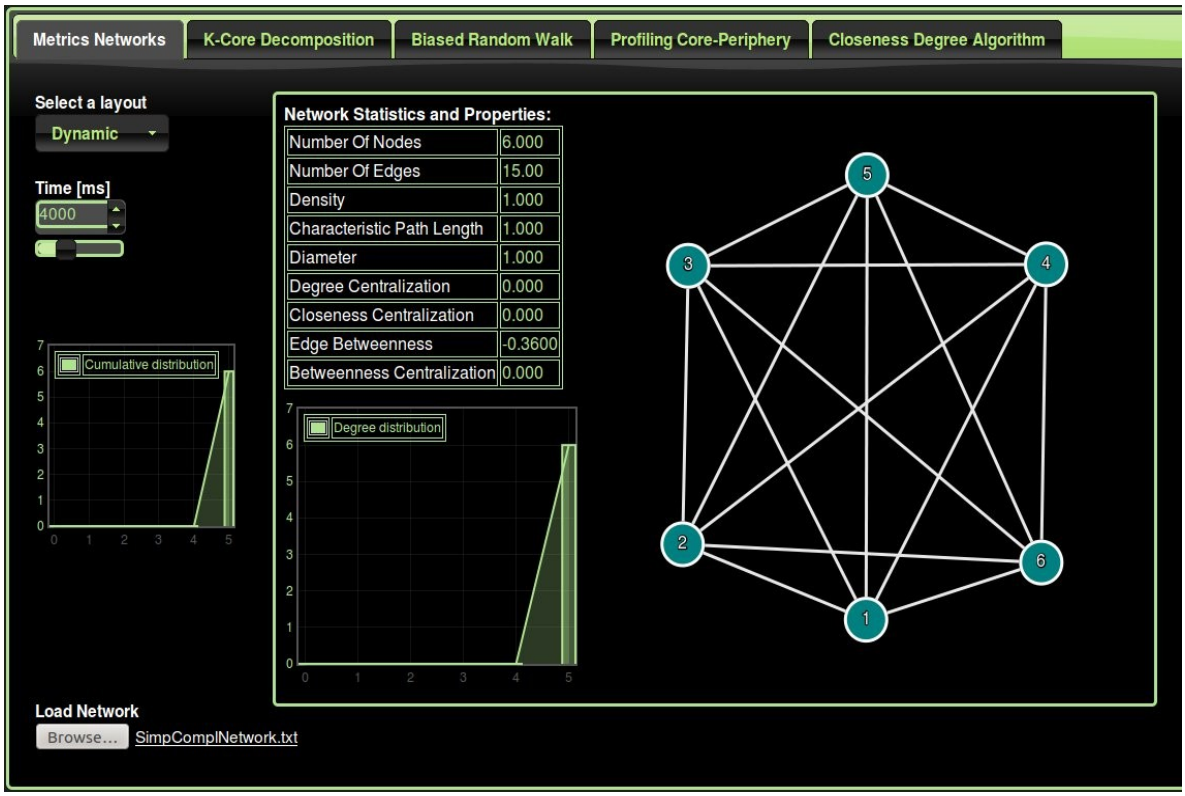


Figure 29: Star: 100% on all measures (degree and closeness) of centralization.

Figure 30: Complete network: 100% on the density measure. Diameter 1.



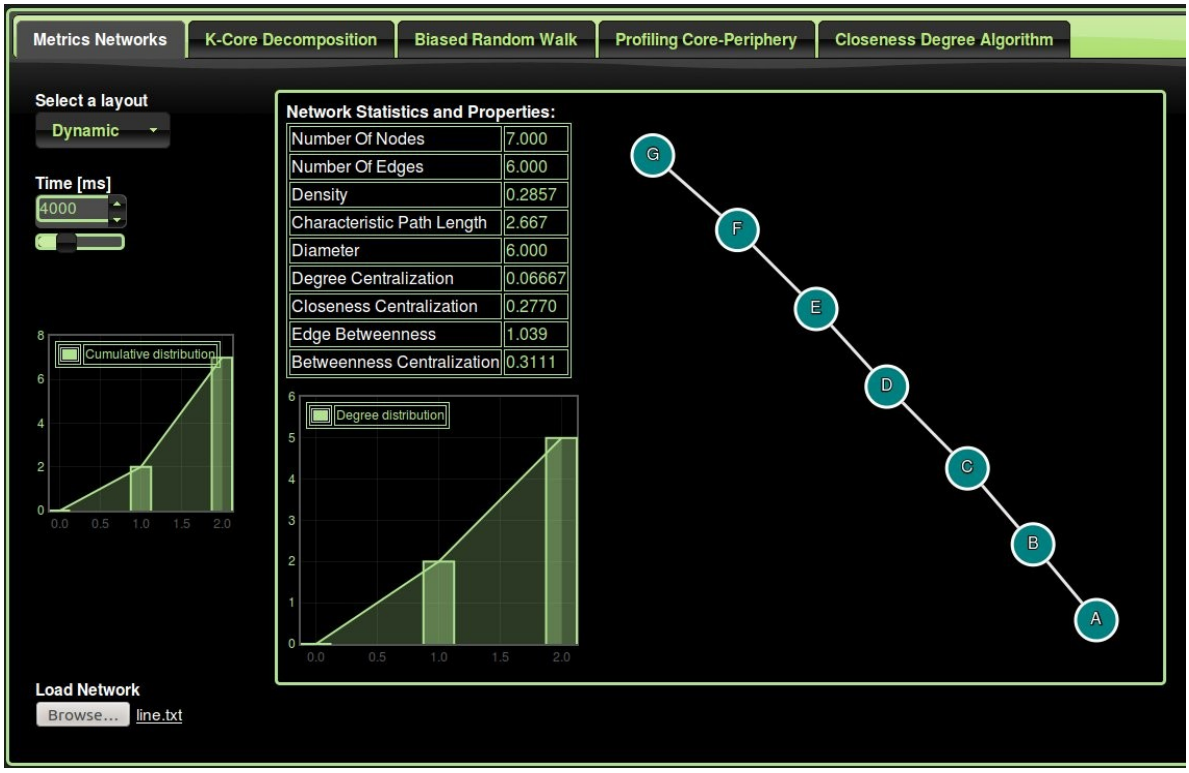


Figure 31: Line network

Appendix 2: Object-Oriented Design UML

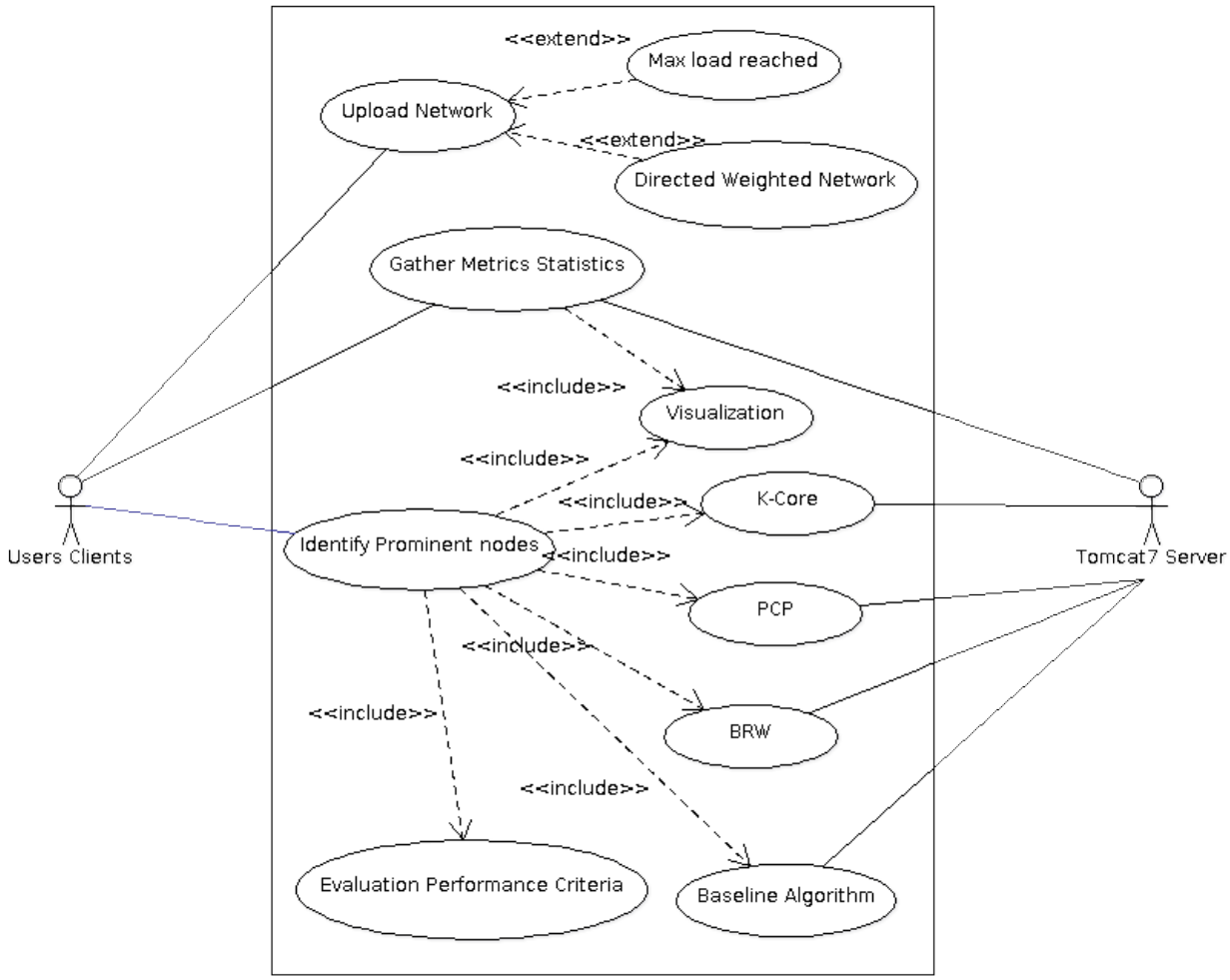


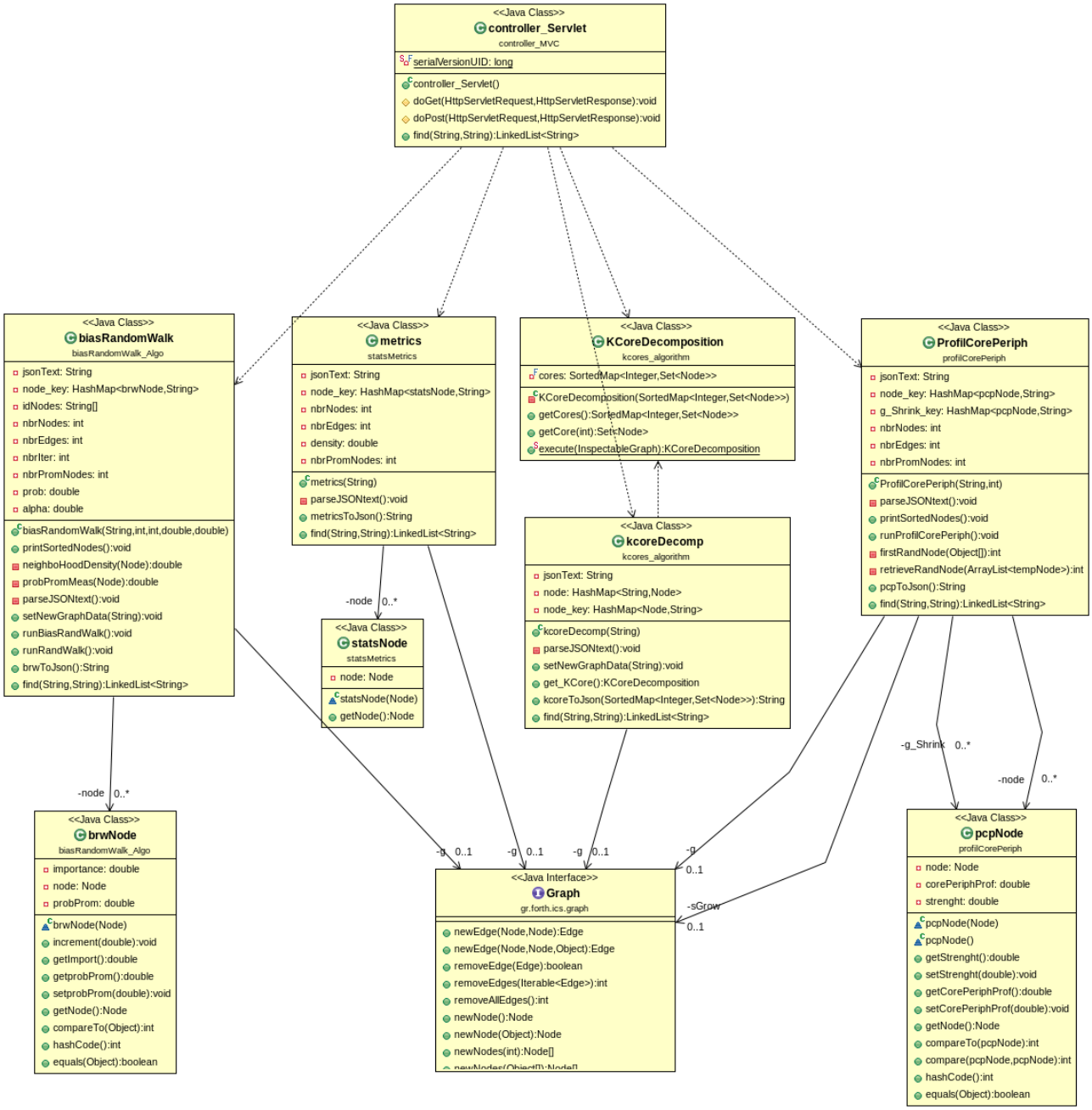
Figure 32: Use Case Diagram of the web application (ArgoUML http://argouml.tigris.org)

K-Core Decomposition Algorithm
BRW Biased Random Walk
PCP Profile Core Periphery

The use case diagram helped us to specify the context and the external interactions with the system. Each use case represents an interaction with the system, named in the ellipse as shown in the previous figure. This diagram also allows to plan iterations of development and validate the system's architecture.

The different steps in the design of our implementation are straight forward once some of the requirements are well defined. First, each use case has been further described, and different scenarios/situations have been performed. Next, a noun and verb analysis and CRC Class Collaboration Cards have been carried out to derive classes, their associated methods. These classes have been grouped into packages, the simplest components, while satisfying any constraint
as shown in the Figure 33.

Figure 33: Class Diagram of some of the classes in the server side.



The Figure 33 shows an overview of the object-oriented design classes and their relationships in the server side. These classes define the objects created dynamically in the system and their interactions. For instance, after receiving the user request, the controller_servlet object can instantiate any object algorithms (metrics, K-Core, PCP, BRW) depending on the request of the user. Then the object algorithm instantiate a Graph object and so on.

Appendix 3: External libraires

JavaScript libraries:

jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

jQuery UI

jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery javaScript Library.

Flot Attractive JavaScript plotting for jQuery

Flot is a pure JavaScript plotting library for jQuery, with a focus on simple usage, attractive looks and interactive features.

Cytoscape

Cytoscape.js is an open-source graph theory library written in JavaScript. You can use Cytoscape.js for graph analysis and visualization.

Cytoscape.js allows you to easily display and manipulate rich, interactive graphs. Because Cytoscape.js allows the user to interact with the graph and the library allows the client to hook into user events, Cytoscape.js is easily integrated into your webapp, especially since Cytoscape.js supports both desktop browsers, like Chrome, and mobile browsers, like on the iPad. Cytoscape.js includes all the gestures you would expect out-of-the-box, including pinch-to-zoom, box selection, panning, et cetera.

Cytoscape.js also has graph analysis in mind: The library contains a slew of useful functions in graph theory. You can use Cytoscape.js headlessly on Node.js to do graph analysis in the terminal or on a web server.

Arbor

Arbor is a graph visualization library built with web workers and jQuery. Rather than trying to be an all-encompassing framework, arbor provides an efficient, force-directed layout algorithm plus abstractions for graph organization and screen refresh handling. It leaves the actual screen-drawing to you. This means you can use it with canvas, **SVG**, or even positioned **HTML** elements; whatever display approach is appropriate for your project and your performance needs.

As a result, the code you write with it can be focused on the things that make your project unique – the graph data and your visual style – rather than spending time on the physics math that makes the layouts possible.

Java Libraries:

flexigraph (Java Graph Algorithm library)

A Java graph algorithms library. This is the old description of the project, briefly touching in few design aspects of it.

Flexigraph is a general framework for graphs. It offers various advantages over most graph frameworks out there (for example, JDSL, JUNG), as for example:

- A much neater API (you will end up writing about half code you would need to write for JDSL to express the same thing).
- Node and edge instances can be shared among many graphs (in JDSL, the same thing would

require to maintain ugly mappings from a logical "identity" to the nodes/edges of any graph, while in JUNG, well, that is just slow).

- Thorough graph event support. For example, in Flexigraph offers a DegreeSorter class which, well, maintains the nodes of a graph sorted on their degree (so you can e.g. process nodes starting with those with smallest degree, etc). If this was implemented in JDSL, one would have to remember every time that the graph is updated, to also re-run the bucket sort. In Flexigraph, you create the DegreeSorter on the graph, and forget! Due to the event model, the graph can be changed and the DegreeSorter remains always consistent with the given graph, with no further ado.
- A better node/edge decoration framework. In other graph frameworks, one can associate arbitrary data to nodes/edges by "put(key, value)" or similar calls. And it is great: it is much faster than having to create a big HashMap containing as keys all the nodes of a graph, for example (try it). So, if one is developing algorithms that operate on a graph and generate data (a simple DFS implementation for instance), he should associate data to nodes/edges this way. All fine and well, except for this being the ideal recipy for creating memory leaks. Because all you, the algorithm implementor, can do is to write a public "cleanup" method to manually erase your algorithm-specific data from nodes/edges (as you don't know when the algorithm's data are not needed any more), and rely on the users remembering to call that method (or else, a memory leak - and good luck debugging that). And worse yet, this depends on the assumption that when the "cleanup" method is called, the graph will be in the state it was when it was decorated with data - cleanup will not be performed on nodes/edges that were accidentally removed before the "cleanup" method.

This is an example of this problem - the DFS implementation of JDSL (notice the cleanup method), or a BFS algorithm of JUNG (notice the removeDecorations method). In Flexigraph, you can either associate strongly data in nodes/edges, like the other frameworks, or associate data weakly. That is, if the key will be garbage-collected (this is possible as it is not referenced strongly in the map inside node/edge), the value will also be automatically cleared. So it is possible to write algorithms that take advantage of node/edge decorations, in a memory-safe way, and with one less thing to remember for all users.

(Extract Source from: https://code.google.com/p/flexigraph/)

JSON-simple

JSON.simple is a simple Java toolkit for JSON objects. You can use JSON.simple to encode or decode JSON text. (source: http://code.google.com/p/json-simple/)

Tomcat7 Server

Apache Tomcat is one of the most popular servlet container, which is available free from the Apache Group at http://tomcat.apache.org. Tomcat can run as a stand-alone servlet container or as part of a web server. There are also several application servers that include servlet containers. For the demonstration of the web application and to set-up multiple clients, I used the GlassFish application server. It is distributed as part of Java EE and included in the NetBeans Integrated Development Environment.

References

[1] Identifying Prominent Actors in Online
Social Networks using Biased Random Walks, Frank W. Takes, Walter A. Kosters Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands, (2011).

[2] k-core decomposition: a tool for the visualization of large scale networks,
Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat and Alessandro Vespignani
South-Paris University, France, Indiana University, USA.

[3] Network Structure and minimum degree, Stephen B. Seidman, Social Networks 5 (1983) 269-287, George Mason University, North-Holland.

[4] Profiling Core-Periphery network structure by random walkers: Fabio Della Rossa, Fabio Dercole, and Carlo Piccardi. Scientific Report 3, Article number: 1467, published 19 March 2013.
http://www.nature.com/srep/2013/130319/srep01467/full/srep01467.html

[5] The map equation: M. Rosvall, D. Axelsson, and C.T Bergstrom
Regular Article. The European Physical Journal Special Topics 178, 13-23 (2009)
Department of Physics, Umea University, 9001 87 Umea Sweden
Department of Biology, University of Washington, Seattle, Wa 98195-1800, USA.

[6] An Information Flow Model for Conflict and Fission in Samll Groups, Wayne W. Zachary, Journal of Anthropological Research, vol. 33, No. 4 (Winter, 1977), pp. 452-473, University of New Mexico.

[7] Identifying the role that individual animals play in their social network, D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations, Behavioral Ecology and Sociobiology 54, 396-405 (2003).

[8] Thurman B. (1979). In the office: Networks and coalitions. Social Networks, 2, 47-63, Syracuse University.

[9] Network of American football games between Division IA colleges during regular season Fall 2000. M. Girvan and M. E. J. Newman, Proc. Natl. Acad. Sci. USA 99, 7821-7826 (2002).

[10] U. Brandes. A Faster Algorithm for Betweennes Centrality. Journal of Mathematical Sociology, 25(2):163-177, 2001.

[11] L. C. Freeman, Centrality in social networks: Conceptual clarification, Social Networks 1 (3) (1978) 215–239.

[12] Large Scale Structure and Dynamics of Complex Networks From Information Technology to Finance and Natural Science Edited by: Guido Caldarelli (University of Rome “La Sapienze”, Italy), Alessandro Vespignani (Indiana University, USA)

[13] S. Wasserman and K. Faust. Social Network Analysis: Methods and Applications. Cambridge University Press, 1994.

[14] A Beginner’s Guide to Graph Theory, ISBN: 978-0-8176-4484-0 , W. D. Wallis, Birkhauser (2000)

[15] Programming the Word Wide Web, Sixth Edition, Pearson, Robert W. Sebesta, ISBN:978-0-13-705383-4, 2011.

[16] JavaScript: The Definitive Guide, O'Reilly Media; 5 edition (27 Aug 2006), ISBN: 978-0596101992

[17] Practical JavaScript, DOM Scripting, and Ajax Project, Springer Verlag GmbH (1 April 2007), ISBN: 978-1590598160.

[18] Murach's JavaScript and jQuery, Mike Murach & Associates Inc. (16 Jan 2013), ISBN: 978-1890774707.

[19] Software Engineering 9, International Edition, Sommerville, ISBN:978-0-13-705346-9, 2011.

[20] Learn Java for Web Development: Modern Java Web Development, Apress, Vishal Layka, ISBN: 978-1430259831

[21] D. Edler and M. Rosvall (2013), The MapEquation software package, available online at http://www.mapequation.org

[22] http://www.cytoscape.org, cytoscape.js, JavaScript library for network visualization, (2014).

[23] flexigraph, a Java graph algorithms library, http://code.google.com/p/flexigraph/

[24] Flot, Attractive JavaScript plotting for jQuery, http://www.flotcharts.org/

[25] Ranking the Importance of Nodes of Complex Networks by the Equivalence Classes Approach, Bojin Zhenga,b,c , Deyi Lic , Guisheng Chenc , Wenhua Dua , Jianmin Wanga, College of Computer Science, South-Central University for Nationalities, Wuhan 430074, China, State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China, School of Software, Tsinghua University, Beijing 100084,China