Visualizing LiveJournal Social Networks Through Clustering

Final Project Report for CS 294-5 (Statistical Natural Language Processing) and IS 247

(Information Visualization and Presentation)

Kirsten Chevalier

December 13, 2005

1 Introduction

This project is an attempt to visualize the social networks associated with blogs on LiveJournal [1]. This is accomplished first through clustering metadata about bloggers based on various features that may be common to many different users. Bloggers on LiveJournal publicly list their "friends", or other LiveJournal users whose blogs they read. They also have profiles that generally contain lists of "interests" (short phrases that they claim to be interested in); then, of course, the full text of their blog postings provide additional data

[1] http://www.livejournal.com/

for clustering. Then, the clustering results are read in by a visualization that displays the graph of friends as a baseline for comparing any further results, and uses color and position to display the additional data gained from performing clustering. There are many different ways to view the friends graph: it can be laid out so as to make connections between friends maximally clear, or in a way that emphasizes the connections between people who have common interests or who write about similar topics in their blogs. How can we combine these different views in order to incorporate as much information as possible into a visualization of the social network, or allow for easy comparison between different possible views? This is the question that this project sets out to explore.

2 Project Goals

This visualization is intended as an exploratory data analysis tool. The ultimate audience would be any user of LiveJournal or anybody interested in analyzing social networks. The tool will be useful for people who want to better understand their social networks, or find new blogs to read based on shared interests or social relationships.

Clustering can remove some of the limitations of straightforward text search when it comes to searching on interests or other metadata, due to the problem of synonyms: for example, if you are looking for someone who lists "books" as an interest (ignoring for the moment the problem of searching through thousands of results), you might miss someone who lists "reading" as an interest, even though listing these two different terms as an interest conveys a similar meaning. On the other hand, clustering might group the person who lists

"reading" as an interest together with other people who lists "books" as an interest – if they list some interests that might co-occur with "books", such as "john irving" or "charles dickens". Thus, clustering can enable a more flexible form of search.

Creating a generally usable visualization tool is beyond the scope of this project, so instead, my goal was to create a visualization that would be useful for me in evaluating my clustering results. My goal was to be able to tune various settings for the clustering algorithms – i.e., which clustering algorithm to use, which similarity function to use for comparing different users to each other, and which features to cluster on – via the visualization UI. I didn't have time to implement interprocess communication between the visualization and the clustering engine (which are implemented as separate applications), so as it is, comparing different types of clustering requires re-running the clustering engine by hand and restarting the visualization, which is awkward. However, the visualization has been useful to me in seeing whether the clustering results correspond to some extent with real social groupings.

3 Related Work

I'm not aware of any previous work in using clustering to analyze blog posts. There have been a few projects that tackled the task of visualizing social networks on LiveJournal and other sites, which I will discuss. In addition, there is some previous work on visualizing clustering results, independently of the domain in which clustering is performed.

3.1 Vizster

Vizster[2] is a visualization whose goal is to visualize the social networks on Friendster, with a specific user as a starting point. Vizster has much in common with this project, with the difference that LiveJournal makes much more data available than Friendster – most importantly, blog posts – providing more data and more challenges for the visualization. Like this project, Vizster provides an interactive graph visualization that allows for viewing details on specific users, and it even has a "clustering" mode that colors subgroups of the user's friends list who are well-connected in terms of friends. However, this project can cluster users based on many different features, and allows for comparison between the different clusterings, which is beyond the scope of Vizster.

3.2 TouchGraph

There are a number of visualizations that simply attempt to visualize the LiveJournal friends network for a particular user, without performing any clustering or other analysis. TouchGraph[3] is fairly representative of such visualizations. It displays a graph based on the friends list for a specific user that includes both the user's friends and their interests as nodes. In addition, clicking on a node for a user "expands" that node, incorporating the friends network of the user represented by that node into the visualization as well. (This is a feature that would be difficult to add to this project, as it would require recomputing the clusterings from scratch.) The problem with straightforwardly displaying the friends graph, as Touch-

[2] Jeffrey Heer, danah boyd. "Vizster: Visualizing Online Social Networks." InfoVis 2005.
[3] `http://www.touchgraph.com/TG_LJ_Browser.html`

Graph does, is that a friends graph as large as even 40 nodes can be difficult to understand without any additional data incorporated into the visualization. TouchGraph does allow the user to hide specific nodes – so that if you're interested in looking at a particular subgroup of your friends network, you can hide nodes that are irrelevant to that group – but with a large network, this can become tedious. Providing clustering data and using it to lay out the graph and color the nodes based on cluster, as this project does, makes the graph easier to understand visually.

3.3 ljviz

ljviz[4] is a text-based visualization that attempts to show friends and interests in different-sized fonts based on popularity among your friends, in the style of extispicious[5], a visualization for user tags on del.icio.us[6]. There are three different modes: friend names can be visualized based on the number of interests they have in common with the given user, interests can be visualized based on the number of friends of the given user who share them, and friend names can be visualized based on the number of friends they have in common with the given user. Obviously, this is a limited range of information given the large amount of metadata available on LiveJournal.

This visualization has major problems with occlusion: if there are several friends who have many interests in common with the given user, the resulting large-font text labels will overlap each other greatly. The labels are drawn with a transparent background, which

[4]http://ponderer.org/lj/ljviz/
[5]http://kevan.org/extispicious
[6]http://del.icio.us

makes the results slightly easier to read, but it's still difficult to see the results in the center. Unlike with a graph-based visualization like Vizster, ljviz also doesn't allow the labels to be moved in order to see what's underneath.

3.4 Hierarchical Clustering Explorer

The Hierarchical Clustering Explorer[7] is a tool for visualizing the results of a hierarchical clustering algorithm, independently of the domain in which the clustering is performed. I find this visualization difficult to understand, and it is probably more useful for scientists than for a broader audience.

4 Clustering Techniques

For this project, I implemented two different clustering algorithms: bottom-up hierarchical clustering, and k-means clustering. Any clustering algorithm could potentially be used in this project, in a modular way, but I ran out of time before implementing more algorithms.

4.1 Hierarchical Clustering

The hierarchical clustering algorithm can be implemented in two ways: bottom-up (agglomerative) or top-down (divisive).[8] In the bottom-up approach, the data set is divided into a list of singleton clusters; then, the algorithm finds the most similar two clusters, and merges

[7] `http://www.cs.umd.edu/hcil/hce/`

[8] Manning, Christopher and Hinrich Schütze, *Foundations of Statistical Natural Language Processing*, section 14.1

them together. This continues until all data elements appear in a single cluster. In the top-down approach, the data set is considered to be a single cluster, then the algorithm finds the least coherent cluster and splits it. Either way, this approach creates a tree of clusters, which is potentially more informative than in the non-hierarchical approach: in the non-hierarchical approach, we know that an element is probably similar to the other elements in its cluster, but we don't know anything about how similar it might be to elements in other clusters. With the hierarchical clustering approach, we can estimate similarity between two elements based on their distance in an inorder walk of the tree of clusters.

Hierarchical clustering requires a notion of similarity between two clusters, as well as a notion of similarity between two data elements, which will be discussed in the "Similarity Metrics" section below.

4.1.1 Experience

Hierarchical clustering produces a list of possible clusterings, where clustering n differs from clustering $n-1$ in that the most similar two clusters in clustering $n-1$ appear merged together as a single clustering in clustering n; hence, for the purposes of the visualization, a single clustering must be picked from somewhere in the middle of this list. The bottom-up hierarchical clustering algorithm had a tendency to keep adding elements to the same cluster, at least at first, so often, the clustering that would be chosen to visualize consisted of one large cluster and several singleton clusters. This is not very useful to the user, so I made a modification to the basic algorithm: at any given step, if k is the size of the smallest cluster that currently exists, then only clusters of size k can be merged together. The exception is if

there is only one cluster of size k. This produces roughly a balanced tree of clusters, which is potentially more useful. An alternative to this solution is to use a complete-link similarity function for comparing two clusters (see "Similarity Functions", below).

After making this modification, there was still a problem, in that there will often be a single cluster of elements that are outliers in the data set, starting with the last two elements that get merged together into a cluster of size 2, and growing from there. A potential solution is to simply filter out users who are outliers (typically this is because these users don't have enough features to eenable any meaningful clustering that includes them) and not display them in the visualization.

4.2 K-Means Clustering

K-means clustering is a non-hierarchical clustering algorithm in which k means are chosen from the data set (where k is a pre-defined constant), then each element is assigned to a cluster that contains the mean that is most similar to it.[9] Then, the mean for each cluster is reassigned to be the element whose average similarity to other elements in the same cluster is greatest. This process is iterated until a fixed point is reached (that is, the list of means does not change), or for a fixed number of iterations (in this project, it runs for 3 iterations, which in practice always seems to be enough to reach a fixed point).

[9] ibid

4.2.1 Experience

The choice of k is an important heuristic when using k-means clustering; either an overly small or overly large choice of k will result in an uninformative clustering. In this project, the implementation of k-means clustering uses a value of k equal to the number of users divided by 4, but it would be desirable to allow the user of the visualization to adjust k and see the clustering that would result for the new value of k. This feature is not currently implemented.

Another important heuristic is the choice of initial means. K-means clustering is very sensitive to this initial choice, as if two elements that are very similar to each other are chosen as initial means, the cluster that should be learned that contains those two elements will be, perhaps arbitrarily, split into two different clusters. Furthermore, if outliers are chosen as initial means, then those clusters may be singletons, and other clusters will be too large. Currently, the implementation of k-means clustering in this project chooses the means randomly. I experimented with choosing the k elements that had the largest number of features as the initial means, but this didn't seem to give much better results than random choosing.

4.3 Clustering Features

Currently, the visualization allows the user to view clustering data based on five different features: interests, friends, tags, links, and LiveJournal users mentioned in posts. I had planned to use n-grams from the full text of posts as clustering features, but didn't have

time to implement this. With tags and links, and with n-grams if this feature were to be implemented, there is a sparsity problem due to technical limitations of LiveJournal that allows only a user's most recent 25 posts to be fetched easily.

4.4 Similarity Metrics

The choice of similarity function is important to the performance of both clustering algorithms. Both k-means and hierarchical clustering require a similarity function that compares two different elements, and hierarchical clustering requires a similarity function that compares two different clusters, as well. In hierarchical clustering, the usual similarity functions between clusters are single-link similarity – which considers two clusters c1 and c2 to be as similar as the most similar pair of elements (e1,e2) where e1 is in c1 and e2 is in c2 – complete-link similarity, which uses the least similar pair instead of the most similar pair, and average-link similarity, which uses the average similarity between members.[10]. I used single-link similarity in my implementation, with the modification described above that compensates for single-link similarity's tendency to create large clusters with high local similarity but low global similarity.

For the similarity function between two different elements, I simply divided the number of features the two elements they have in common by the total number of features the two elements have. It would probably be better to weight the features in terms of popularity, discounting more popular features as they give less information about the data, but in

[10]ibid

practice, the features that were used for clustering in the example data sets I used were never very popular, so I did not experiment this.

It would be desirable to extend the visualization to allow for experimenting with different similarity functions and comparing the results based on using different similarity functions.

5 Evaluating the Clustering Results

I did not have time to do a usability study of the visualization itself, but as its usefulness is largely dependent on the usefulness of the clustering data, I did an informal study of how useful the clustering data was, to determine whether the visualization had potential to actually provide insight on the problem. Since this is largely determined by subjective evaluation, I solicited ten LiveJournal users and asked them to read over a list of clusters based on their friends list and comment on whether those lists bore any relation to real social groupings. Eight users responded.

Each user was shown a randomly selected list of clusters that were drawn from the results of four different clustering methods: hierarchical clustering based on interests, hierarchical based on friends, k-means based on interests, and k-means based on friends. They were not told what features were being used to do the clusterings. Most users agreed that the clusterings based on friends were more coherent, and a smaller majority found the k-means friends clusterings to be more coherent than the hierarchical friends clusterings. Users generally were confused by the results of the interests-based clusterings, and didn't understand why users who were grouped together were grouped together. This is somewhat disappoint-

ing, since there are already tools for visualizing friends groups that work well without using clustering, but on the other hand, the details view that the visualization provides might be useful for understanding clusters based on non-friends features: if a user doesn't understand the basis behind a cluster, they can see exactly which features users in that cluster have in common.

6 Visualization

The visualization is written in Java, using the prefuse[11] toolkit. The input is a set of XML files generated by the clustering engine, which is written in Haskell. The central component of the visualization is a graph showing the LiveJournal friends network of the user represented by the given data, in which the nodes are colored based on the cluster to which the users belong. On the right side of the window, two details panes appear: the top one is refreshed when the user mouses over a node, and it shows the features that users in the node's cluster have in common, sorted by popularity. The bottom pane shows details for the node being moused over, including the username corresponding to the node, a user picture for the user, and a list of features that the user has in common with other users in their cluster. The window also includes a text box in which the name of a feature can be typed; any users who share that feature will be highlighted in a distinct color.

The feature to be clustered on can be selected using a row of radio buttons at the bottom of the window: currently, the features supported are friends, interests, tags (users

[11] http://prefuse.sourceforge.net/

can apply tags to individual posts), links contained within posts, and other LiveJournal users mentioned in posts (these can be referenced in LiveJournal using a special HTML tag, which allows them to be identified more easily). Currently, there is no support for combining different clustering features in the same view. When switching between features, the underlying graph visualization provides somewhat of an "animation" effect: changing between features is implemented by adding and removing invisible edges between nodes in the same cluster, so when these edges are added and removed, the nodes gradually move in order to assume their new layout, and this provides a sort of dynamic view of the differences between clusterings based on different features.

Additionally, right-clicking on a node brings up a menu with two options: "Hide This User" and "Highlight This User". Hiding a node makes it invisible; all hidden nodes can be restored by clicking the "Unhide All" button on the left side of the window. Highlighting a user allows for displaying the features that an arbitrary set of users have in common. When highlighted users exist, the list of currently highlighted users appears in the lower right-hand details pane, along with a list of features they have in common. This can be undone by right-clicking on any node and selecting "Unhide All".

Figure 1 shows the application on a sample friends graph, clustering by friends, after mousing over the user "premaloka".

Figure 2 shows the application on the same graph, clustered by interests, after highlighting the users "neuracnu" and "lcremeans" and mousing over the user "neuracnu". The upper right-hand pane shows all the interests shared by users in the cluster that includes

these two users; the lower right-hand pane shows the interests that these two specific users have in common.

Figure 3 shows the application on the same graph, clustered by LiveJournal users mentioned in posts, after typing "shelley42" into the search box and mousing over the user "wellstar". Notice that the nodes "captain18", "wellstar", "scouttle", and "miang", all towards the center right edge of the screen, are displayed in bright yellow. These are all users who mentioned the LiveJournal user "shelley42". Also notice that some nodes are shown in bright red; these are users who lack any instance of the current feature, in this case, users who don't mention any LiveJournal users in recent posts.

6.1 Data

The clustering engine fetches XML data from LiveJournal, using a special file provided by LiveJournal for each user to determine friends and interests data, and using the RSS feed for each user's blog to get data based on the full text of their blog posts, such as tags and links. A limitation of this approach is that the RSS feed only provides the last 25 posts, so when clustering based on tags or links, the data is quite sparse.

The clustering engine takes a username as an argument and currently only clusters users on that user's friends list. I originally planned for it to use the list of the user's friends and all friends listed by their friends. However, I ran into performance problems with the Haskell XML parser when using a dataset of this size (for example, a user with 90 friends might have 6200 friends-of-friends). This would potentially make the visualization significantly

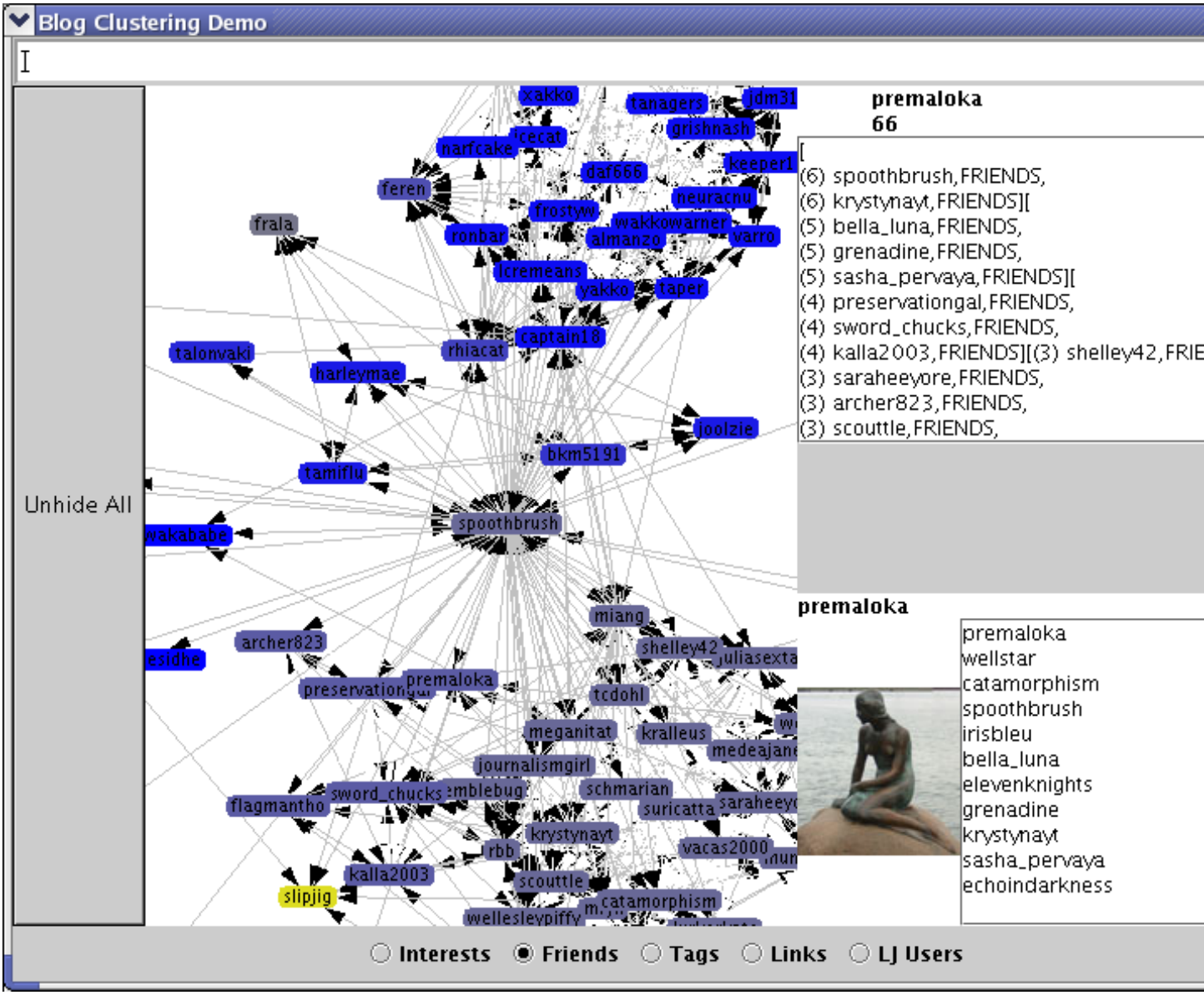

Figure 1: The application on a sample friends graph, after mousing over the user "premaloka"

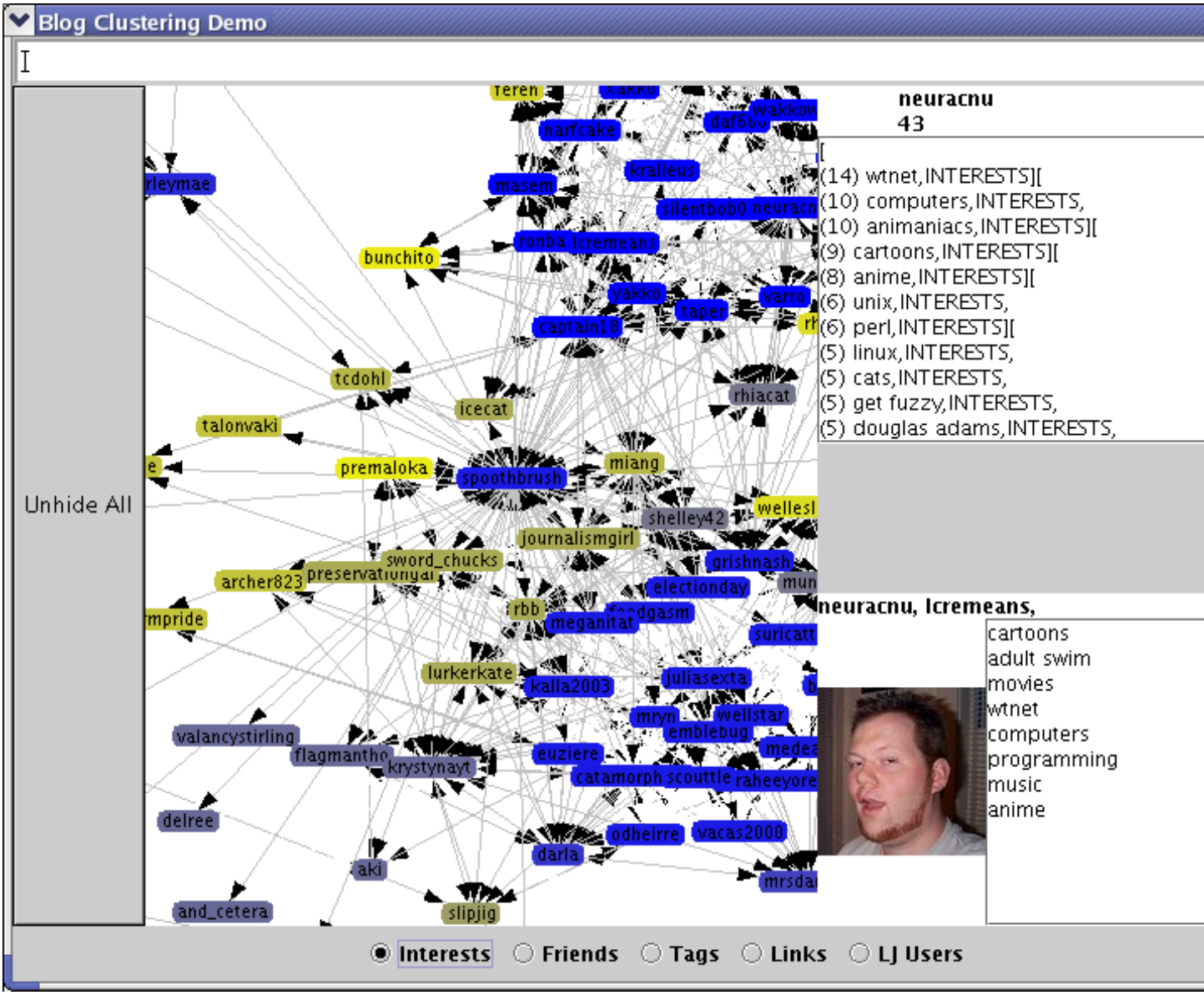

Figure 2: The application on a sample friends graph, after highlighting the users "neuracnu" and "lcremeans" and mousing over the user "neuracnu"

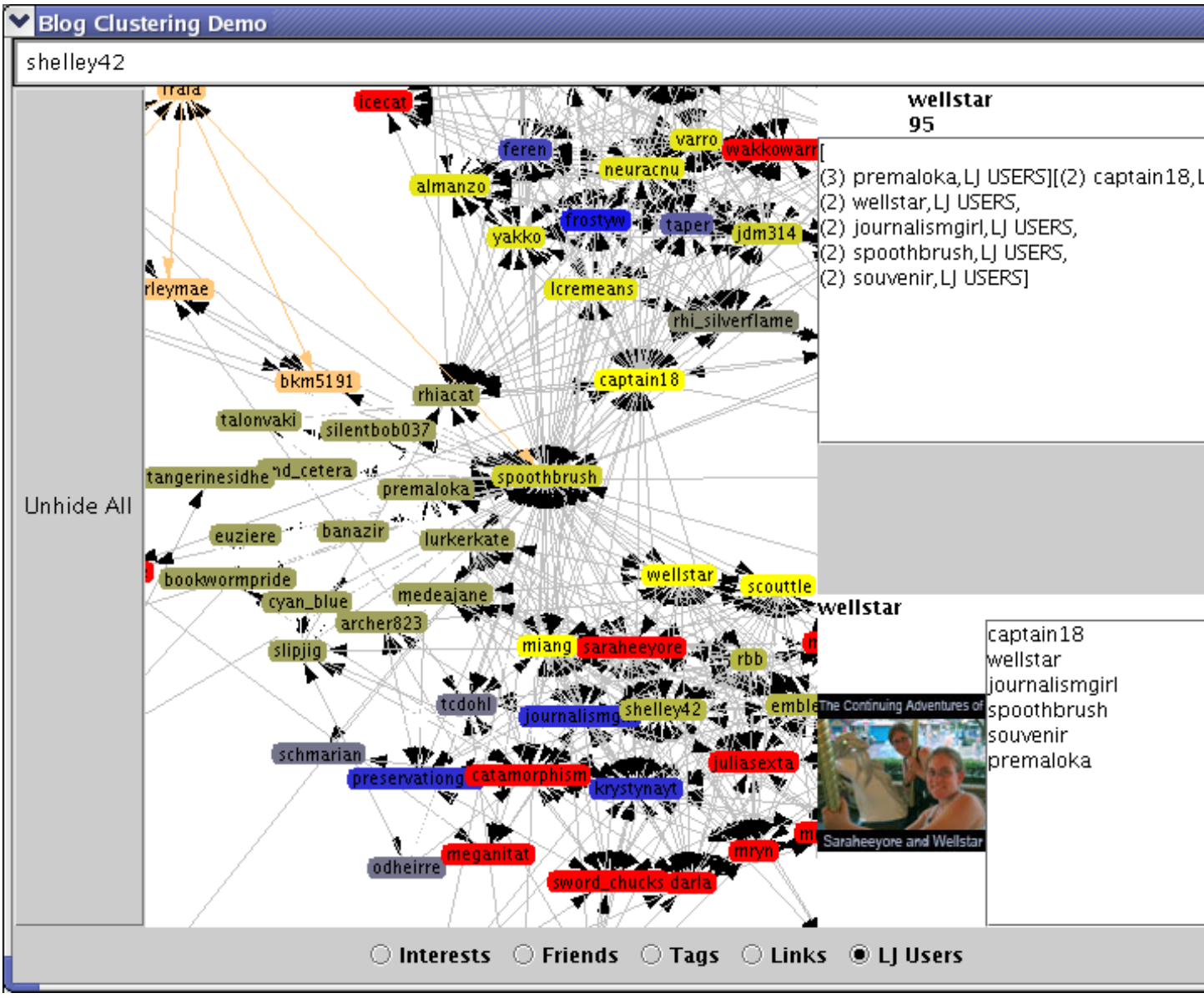

Figure 3: The application on a sample friends graph, after typing "shelley42" into the search box.

more useful – as one is likely to be already familiar with the characteristics of users on one's own friends list – but I've left it for future work.

The clustering engine currently implements two different clustering algorithms – hierarchical bottom-up, and k-means – and one desirable task for the visualization would be to compare results from different algorithms. This is not currently implemented, and instead, the k-means data are used.

6.2 Use of Visual Properties

The visualization uses the property of color to denote which nodes are in the same cluster. The colors are computed by dividing a range of possible colors into n colors, where n is the number of clusters. This can potentially result in using colors that are hard to distinguish, but since n is typically small, this typically won't be a problem. In addition, the property of which cluster a node is in is redundantly indicated by spatial position. As described above, edges are added and removed as needed between nodes that are in the same cluster (based on the currently selected feature). The visualization uses Prefuse's force-directed layouts, so nodes are drawn closer to nodes they are connected to, and clusters end up more or less corresponding to the spatial position of the nodes.

In addition, color is used when searching on a specific feature: matching nodes are displayed in a bright yellow color which is distinct from the other colors used.

7 Future Work

In this section, I will describe a number of visualization features that I would have liked to implement, but didn't have time for.

7.1 Larger Networks

As mentioned above, I was originally planning for the visualization to be used to visualize a user's friends of friends, rather than just friends. If the performance problems in the back-end are solved, then there will still be issues to address within the visualization for visualizing larger networks. The graph will certainly not fit entirely on one screen, but since the visualization includes a panning tool, and since users in the same cluster will be drawn close together, this may not be a problem.

7.2 Interprocess Communication

I would like to add controls to the user interface to change clustering features such as the algorithm type, the similarity function, and the method of choosing initial means (for k-means clustering), so that I can twiddle these knobs and watch how the clusterings dynamically change as a result. These would send a signal to the clustering back-end (which would run in the background while the visualization runs, unlike now, with the clustering back-end communicating with the visualization front-end via files) to generate a new clustering based on the new parameters.

7.3 Visualizing the Hierarchy of Clusters

Though the visualization currently doesn't use data from hierarchical clustering at all, it would be idea to display the tree of clusters that hierarchical clustering creates, overlaid with the friends graph. The alternative would be to pick a particular level at which to take a slice of the tree, which gives a single set of clusters, but as hierarchical clustering provides more information than non-hierarchical clustering, it would be ideal to take advantage of that information.

7.4 Indicating Confidence in Clusters

The back-end could potentially be modified to output data about how much confidence it has each cluster: a measure of confidence could be the number of users in the cluster who share the most common feature in the cluster, divided by the number of the users in the cluster, and other measures are possible. Then, the visualization could use color or text labels to show how "good" a cluster is, i.e., how much confidence the clustering engine had in it. In addition, it could provide a way for the user to filter out clusters whose confidence falls below a certain threshold, which would be very useful when displaying larger networks with many clusters. It would also be useful to allow clusters containing only one user to be filtered out, since these clusters provide no information.

7.5 Showing Overview Data

Though not related to the goal of visualizing clustering, users might find it useful for the visualization to display an overall list of which features were most popular within their friends networks.

8 Links to the code and thumbnail

All the code for this project is available at `http://www.sims.berkeley.edu/courses/is247/f05/projects/cblog/InfovizCode.tar.gz`. After unpacking the tarball, two directories will appear: `ClusteringCode`, containing the Haskell code for the clustering backend, and `prefuse-alpha-20050401`, which is a Prefuse distribution that contains my original code in the subdirectory `KirstenProject`.

Building the Haskell code requires ghc to be installed. The shell script `m.sh` within the `ClusteringCode` directory builds the code. The code expects the directories `/ProjCache/prefuseClusterI` and `/ProjCache/prefuseGraphs` to exist. The XML files for the clustering results and friends graph, respectively, will be saved in those directories, with names corresponding to the name of the user for which the data was generated.

The Java code expects the data to appear in the `etc` subdirectory of the Prefuse root directory, as `user.xml` for the friends graph and `user_clusters.xml` for the clustering files. To make things easier, I have included data for four users inside the `etc` directory in the tarball, so that the Java code can be run without running the Haskell code. These users are jholomorphic (a small graph), rbb (a medium-sized graph), spoothbrush (a large graph),

and catamorphism (a slightly larger graph). The user on which to run the code is specified at the very beginning of the `main()` function in `KirstenProject.java`.

Due to technicalities, the user icons shown in the right-hand corner in the screenshots will not appear when the code is run elsewhere.

All files, including a thumbnail image, are linked from `http://www.sims.berkeley.edu/courses/is247/f05/projects/cblog/`.