Visualizing Coclustered Matrices

David Watkins Columbia University
New York, New York USA
January 4th, 2016
djw2146@columbia.edu

ABSTRACT

Many videos on the Web about international events are maintained in different countries, and some come with text descriptions from different cultural points of view. We perform a spectral decomposition algorithm to cluster these videos based on their visual memes and their written tag identifiers. The spectral decomposition provides a matrix containing tags clustered with tags, and coclustered with visual memes, as well as visual memes clustered with visual memes and coclustered with tags. We take one of these coclustered matrices and provide a Web service for visualizing the clustering in scatterplot format, force-directed graph layout, and histograms. In addition we have demonstrated that Applying algorithms such as Reverse Cuthill McKee can allow for the viewer to see a diagonalized representation of the matrix.

1. MOTIVATION

In coordination with Professor John Kender at Columbia University and PhD candidate Chun-Yu Tsai at Columbia University, the current research project, *Cross-Cultural Annotation for Video Archives of Human-Interest International Events*[5], generates a coclustered matrix with visual memes and tags as the axes. The original method for visualizing the coclustered matrix was to use the matrix display functionality within Matlab. This grew cumbersome and was not representative of the vision for the planned consumer web application. Thus a new web based data visualization tool was conceived utilizing new web technologies and additional functionality not present in Matlab.

2. METHODS

In the next few subsections I will discuss the technologies and algorithms used in the process of the research project.

2.1 Background

Throughout this discussion of my research I will frequently refer to *visual memes* and *tags*. Visual memes correspond to frames of a video that have been analyzed and cataloged previously. Each of these frames have a series of textual tags associated with them. These tags and frames are then independently and co-clustered to generate a large matrix. One of the quadrants of this matrix has a section devoted to visual memes in the y-axis and tags in the x-axis, which is the basis for the data used in this project.

2.2 Algorithms

2.2.1 Compressed Row Storage Matrix

When using sparse matrices, frequently a compressed row storage matrix (CRS) is used. The matrix is stored using two vectors, the *column-index* corresponding to the values that are set to 1, and the *row-pointer* corresponding to the value indexes where each row starts. For example consider the non-square matrix

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The number of rows is 3 and the number of columns is 4. The number of rows and columns must be stored to make sure the original matrix can be recreated later. The *row-pointer* vector will correspond to the values $\langle 1, 4, 6, 8 \rangle$ and the *column-index* vector will correspond to the values $\langle 1, 3, 4, 1, 2, 3, 4 \rangle$[3]. This example assumes that matrix only had values of 1 and 0. There is a more general algorithm that allows for varying values.

CRS was used for my research project specifically because the Reverse Cuthill McKee implementation I was using relied on the Python library Numpy. In its implementation it assumed the matrix being passed was in CRS format and so I implemented functionality that converted the dense matrix between CRS and dense formats in Javascript.

2.2.2 Reverse Cuthill-McKee

The Reverse Cuthill-McKee algorithm (RCM) is an algorithm to permute a sparse matrix that has a symmetric sparsity pattern into an organized matrix with a smaller bandwith. The algorithm produces an ordered *n-tuple* of indexes which is the new order of the vertices[2]. The algorithm requires an $n \times n$ matrix.

Because the matrix I was provided did not have the characteristic $n \times n$ that was required, I performed the multiplication of the tranpose with the original matrix to generate a matrix of the appropriate dimensions. This did mean that any block diagonalization of one dimension effectively ignored the other because the data corresponding to the other axis was removed. In order to reorder the visual memes, AA^T was used. In order to reorder the tags, A^TA was used.

The entirety of the code listing for RCM is based on a python implementation built using Numpy[1]. The specifications of the Python implementation are not important, but all functionality that existed within the numpy library was implemented correctly for Javascript. In particular the matrix functions for multiplication and transpose were implemented.

Argsort was also implemented in Javascript which sorts an array and returns an array of the indexes that correspond to the sorted positions for each of those items.

2.2.3 Sørensen-Dice Coefficient

The Sørensen-Dice Coefficient (dice filter) is a statistic used to compare the similarity between two vectors. It is most commonly calcualted for two vectors as the following:

$$d = 2|A\dot{B}||A|^2 + |B|^2$$

The value d corresponds to a specific coefficient relating the similarity of the two vectors[4].

I used this statistic to determine whether two columns had enough similarity that they could be effectively combined. I would then remove one of the columns and then continue filtering the rest of the matrix. I only implemented column comparison but adding the collapsing of rows would also be something worth evaluating. The specific dice coefficient threshold is a simple parameter defined within the UI on the application.

2.2.4 *Histogram Filter*

A histogram filter counts the number of occurrences at a particular row or column and then removes the most and least common of the entire histogram generated. Due to the density of the data used a cutoff of 60 most/least common was used to remove data from visualization. This value is very dependent on the data used as well as the sparsity, and therefore would need further research to determine a robust way of generating such a value.

2.3 Javascript

The majority of this project relies on various Javascript frameworks including Angular.js and the Angular Material Design framework, vivagraph.js, d3.js, and highcharts. Research for the best frameworks to use were performed over the course of the semester as new features were conceived or problems arose from the existing frameworks.

2.3.1 *Angular.js and Angular Material Design*

Angular.js is a 100% Javascript framework that extends basic html functionality to offer Model-View-Controller (MVC) design principles within the browser[1]. In the context of this project Angular allowed me to build simple communications with the web server in the form of services that would specifically load the files needed to display all of the graph data. Getting each file is broken down into a single function call each.

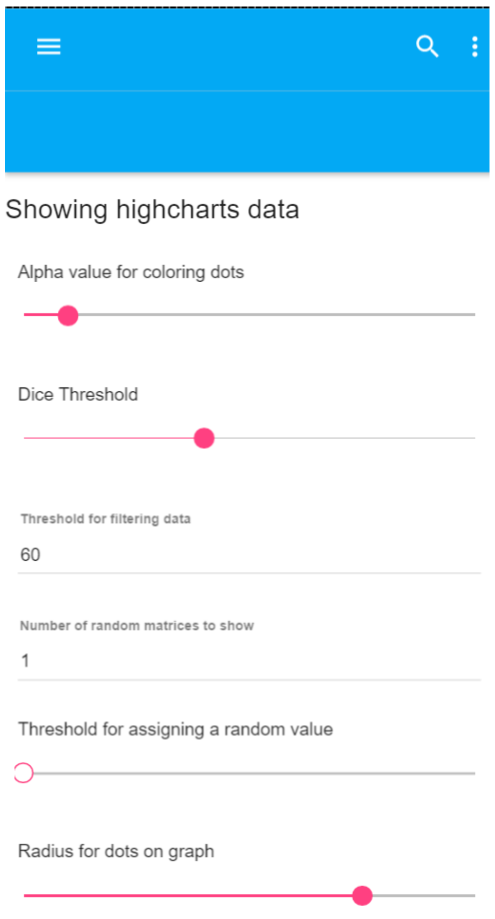
```
service.getMatrix = function($scope, id, q) {
        $http.get(matrixURL).then(function(response)
        ↪  {$scope[id] = response.data;
        ↪  q.resolve();});
}

service.getVisualMemeIndex = function($scope, id,
↪  q) {
        $http.get(visual_meme_indexURL)
        .then(function(response) {$scope[id] =
        ↪  response.data; q.resolve();});
}

service.getCluster1 = function($scope, id, q) {
        $http.get(cluster1URL)
        .then(function(response) {$scope[id] =
        ↪  response.data; q.resolve();});
}
```

Listing 1: Loading 3 files easily using Angular services

The application for visualizing the data was created simply by loading one web page and then embedding additional html snippets into a section designated within the html code. This allows for a more responsive UI as opposed to loading each individual data visualization as a separate UI. As mentioned before the html templates were written using Jade thus making the code look less like html. There was no conflict using Jade and Angular together in the same application.

Angular Material Design (AMD) is a Javascript and css framework that offers simple to use functionality for building attractive and user friendly user interfaces[2]. It works very well with Angular.js and thus was a simple choice to use in this research project. While bootstrap was used in the previous version of the UI, it is not designed for use with Angular and therefore would have slowed down development of the application. Use of AMD also lets the user interface to be easily accessible from a mobile browser without further configuration.

[1]http://Angularjs.org
[2]http://Material.angularjs.org


Showing highcharts data

Alpha value for coloring dots

Dice Threshold

Threshold for filtering data
60

Number of random matrices to show
1

Threshold for assigning a random value

Radius for dots on graph

Figure 1: Ease of use of new interface in mobile setting

Angular also allows convenient creation of additional controllers. Through defining a controller for each view I was able to make sure the code I wrote was concise and only served one purpose.

```
app.controller('highchartController',
function highchartController($scope, $q,
↪  loadService, parseService) {
...
}
```

Listing 2: Defining the highChartController in highChartController.js

The $\$scope$ variable within an angular application defines what each controller has access to within its own scope. It is an effective way to make sure controllers are self contained entities. The $\$q$ parameter is used in angular to allow for the application of promises, or bits of code that need to be executed in a specific order[3]. The additional two services that are passed as parameters were custom written specifically for this application. For more information about them see section 2.4.

Angular works very well with other frameworks so long as they do not modify the HTML code of the web page. Angular makes frequent modifications to the Domain Object Model (DOM) of a web page as part of its design, so when a controller reloads part of a web page could interfere with another framework that assumes the region is still there. The use of d3.js and Highcharts were specifically chosen because they operate agnostic of the current DOM and will reload any number of times. However performance is certainly a factor that was not optimized during this research process.

2.3.2 Vivagraph

Vivagraph is a Javascript library written for drawing force directed layouts[4]. A force directed layout is specifically used for drawing large dynamic graphs without specifying the absolute positions for the nodes. My initial approach to supporting this layout was to allow for visualization of how each of the connections between either visual meme nodes, tag nodes, or both visual memes and tags existed in the data. The use of Vivagraph initially was because it appeared to be the fastest Javascript framework for visualizing force directed graphs.

Through initial testing the first graphs generated by the framework were subsets of the data so as to demo the application of a force directed layout. As more of the data set was added to the visualization it slowed the browser so much that it was difficult to close the application. In this instance I had loaded 75% of the matrix by simply capping the amount of rows and columns I added to the graph layout at 75%. Also in this case only the visual memes were the nodes and the tags were the connections between each visual meme.

I had also attempted to use both tags and visual memes as nodes and each link was a location in the matrix that was set to 1 for that visual meme index and tag index. This was with 40% of the data from the matrix.

After some discussion based on these initial findings, it was decided that the best course of action would be to intelligently reduce the number of nodes being viewed by the force directed graph. The first initial attempt was to use a histogram filter on the data. Because of the density of the graph, the 40 most common and 40 least common nodes were removed from the graph. Otherwise the new view of the graph would be non-performant.

During development I tried to incorporate colored nodes to denote whether it was a US video, European video, neither or both, however Vivagraph did not offer a convenient way of doing so. I also experimented with using quadrants of data to reduce the number of nodes. In this schema each node was roughly a 40×40 submatrix of the original matrix. The links between each of the

[3]https://docs.angularjs.org/api/ng/service/$q
[4]https://github.com/anvaka/VivaGraphJS

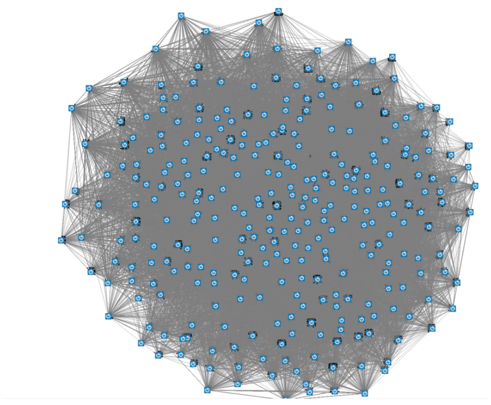


Figure 2: Vivagraph force directed graph at 75% of the data matrix

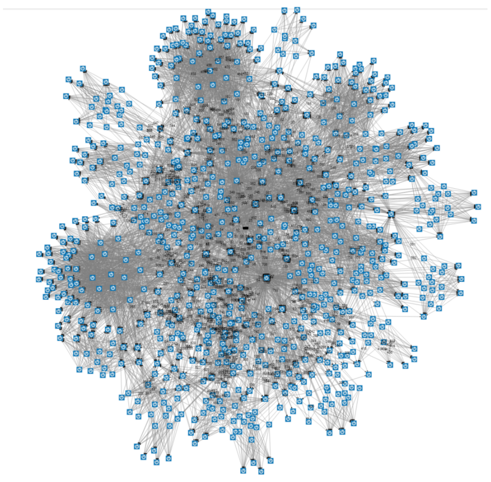


Figure 3: Vivagraph force directed graph at 40% of the data matrix

nodes were a situation in which any of the inner nodes shared a tag with another submatrix.

2.3.3 D3

As I kept developing I realized that Vivagraph was becoming too slow of a solution to offer real-time support for force directed graph visualization. My options were to remove force directed graphs from the visualization tool, produce static images on the server side, or find another Javascript based framework. d3.js is a very popular data visualization Javascript framework that has been in active development since 2011. d3 supports many forms of data visualization, including force directed graph layouts[5].

I followed this tutorial in order to initialize d3 in the browser and display a force directed graph. d3 makes it easy to modify graph link strength and associate colors with nodes. It also has additional functionality for supporting mouseover events to see when a specific node is being highlighted by the user. These features made it simple and easy to color nodes based on their types (specifically US Videos and European videos) and also made it easy to represent how connected two nodes were.

[5]http://d3js.org/

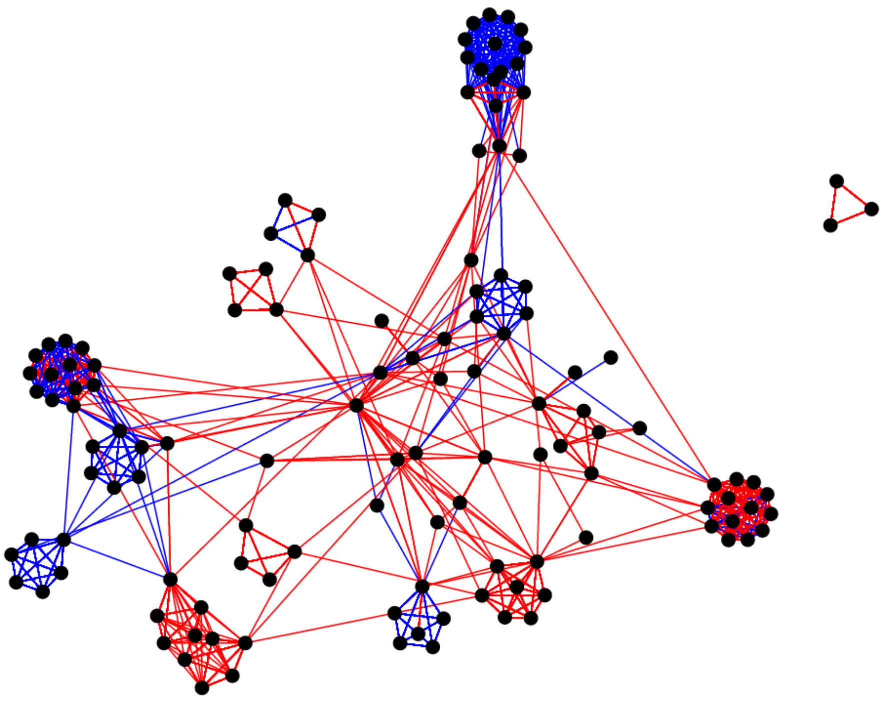


Figure 6: d3 force directed graph with tags as nodes and visual memes as links

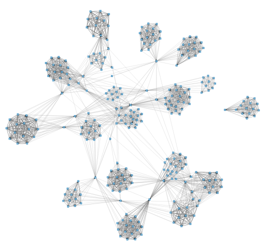


Figure 4: Vivagraph force directed graph with histogram filter

I experimented with the use of visual memes as nodes, tags as nodes, and both tags and visual memes as nodes with d3 in particular. I have offered these views as a setting in the UI as well (for more information see the tutorial). In these three scenarios I had to use histogram filtering in order to reduce the number of nodes to a reasonable level to be viewed. The default value set was 60, meaning that the 60 most and least common nodes were removed from the view. The most interesting observations made during this process is that clear relationships could be determined between the clusters of nodes. The higher the number of connections between any node and another node increased the thickness of the links. For my testing I used red nodes to convey US Videos, blue nodes to convey European videos, purple nodes to convey videos corresponding to both cultures, and grey nodes to convey videos that belonged to neither.

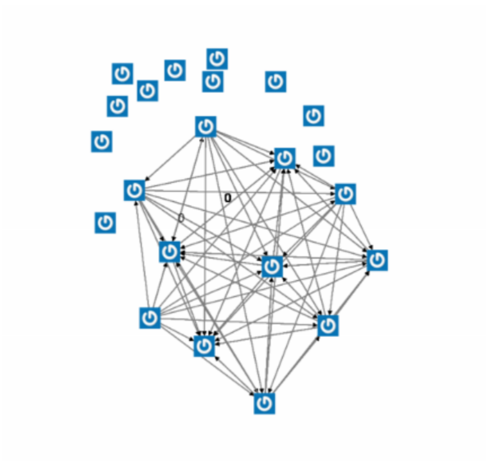


Figure 5: Vivagraph quadrants of 40

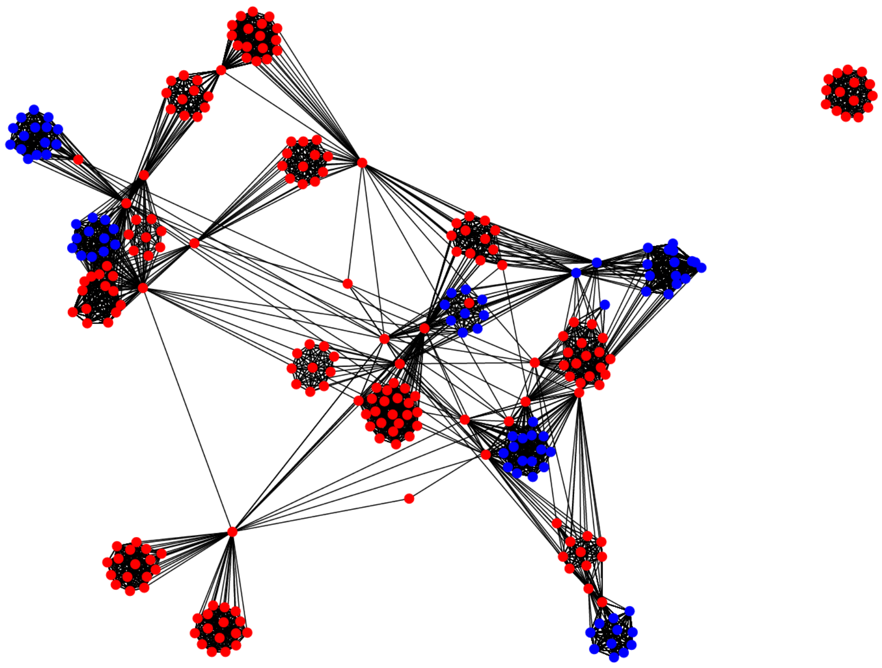


Figure 7: d3 force directed graph with tags as links and visual memes as nodes

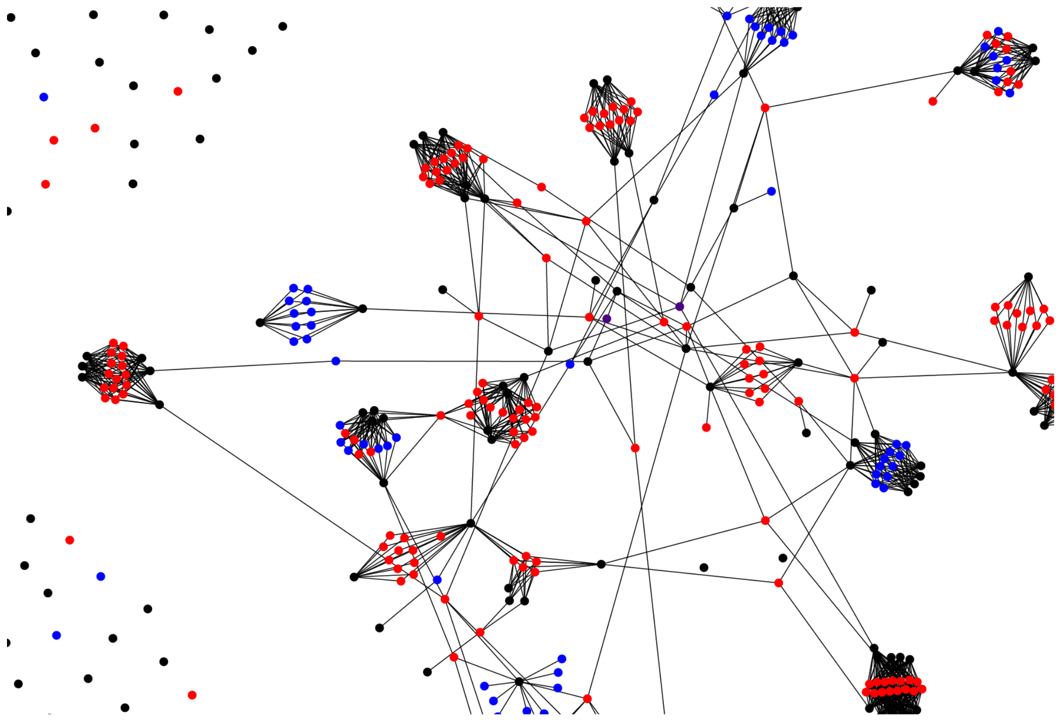


Figure 8: d3 force directed graph with tags and visual memes as nodes

2.3.4 Highcharts

To display the matrix in the simplest way possible, a scatterplot graph was used. After researching many solutions, Highcharts appeared to be the most feature rich while also being performant for excess of 1,000,000 data points. Highcharts has support for coloring nodes and uses sparse storage of data points to allow for dynamic data display[6]. Using Highcharts I was able to quickly iterate on features we discussed in our meetings.

Using the Highcharts graph I was able to conveniently display data filtering techniques such as Dice Filters and Histogram Filters and also reorganize data using the Reverse Cuthill McKee algorithm. I also implemented additional functionality to approximate additional matrices based on the first one by adding random noise to the graph. The rate at which random noise and the number of additional matrices generated were both parameters built in the UI. In order to visualize change over time, or view multiple matrices simultaneously the alpha value for the color of each of the nodes was reduced below 1 and each generated matrix was added to the high charts graph.

For the block diagonalization I implemented a version of the Reverse Cuthill McKee algorithm that relied on compressed row storage of the matrix that would reorganize both the tag axis and the visual meme axis. I was able to use AMD to quickly add a setting to allow the user to choose either of these organizations or both at the same time.

Also built using the HighCharts library were a series of Histograms that allowed for viewing how common a particular visual meme or tag was in the matrix. This was constructed very easily by iterating over each of the axes of the matrix. These histograms were then used again for the histogram filters.
In addition to the histogram filters I implemented the dice filtering algorithm to reduce the number of rows and columns that meet a specific similarity determined by the dice coefficient of two columns. I only implemented column comparison, however adding additional functionality for comparing rows would be trivial. The user is able to adjust the dice coefficient using a slider on the UI.

2.4 Pre-existing Code

Provided by Skylar Pagenkopf, a wealth of pre-existing codebase for a user interface to display cross cultural news videos and their information. The application relied on Node.js[7] for serving the content, MongoDB[8] for storing the data associated with the videos, and Jade for templating the web pages. In the user interface it featured web frameworks such as Bootstrap[9] and Timeline.js[10].

For the purposes of this project, only the Node.js backend and the jade templates were modified from the original source to allow support of the new data visualization tools. Additional source files were added to the project which included Javascript frameworks, additional jade templates, and matrix files. The login functionality was disabled to make sure that development was simpler and because the data visualization tool did not rely on data that was secured by user access.

2.5 New Code Layout

For this application several new files were added to the existing source. In order to most conveniently serve the content for this application, the majority of the files used were placed in the "public/js" folder. The following Javascript files were added:

2.5.1 appController.js

Contains the listing for each of the views (force directed graph, scatterplot, and histogram) and defines a simple controller to populate the list of navigable views.

2.5.2 graphCleanerService.js

Contains several functions for modifying and manipulating the matrix retrieved from the server. Used by all three of the views for filtering and applying the RCM algorithm to the data. This Javascript file was built using the Angular service functionality.

- **calculate_data** - Generates a series of fake matrices based on the initial one.
- **generateHistogram** - Takes in a matrix and generates the corresponding histogram for a particular axis.
- **filterData** - Filters the data using histogram filters.
- **parseData** - Converts the textual representation of the matrix (A.txt) and converts it to a 2D array.
- **parseMemeIndex** - Converts the textual representation of the visual meme index and converts it into a dictionary for easy lookup of each visual meme index.
- **parsecluster** - Converts the textual representation of the cluster file (Cluster1.txt) into a dictionary to lookup the cluster index and determine the cultural origin of the visual meme.
- **dice_coefficient** - takes two arrays and computes their dice coefficient.
- **rcm** - Takes a matrix and returns the RCM block diagonalized form of it.

2.5.3 highChartController.js

Controls the scatterplot view of the matrix. Built using the Angular controller definition protocol. Utilizes both the loadDataService and the graphCleanerService to manipulate data.

2.5.4 histogramController.js

Controls the histogram view of the matrix. Built using the Angular controller definition protocol. Utilizes both the loadDataService and the graphCleanerService to manipulate data.

2.5.5 loadDataService.js

Uses the Angular *$http*[11] directive for retrieving the files from the web server. Built using the Angular protocol for services.

2.5.6 vivagraphyController.js

Controls the force directed layout view of the matrix. Built using the Angular controller definition protocol. Utilizes both the loadDataService and the graphCleanerService to manipulate data.

In order to make sure the styles of the data visualization tool were correct, a *display.css* file was added to "public/css" for this page only. The following jade templates were added to the directory "views":

2.5.7 display.jade

Controlled by the appController, this view contains the template for all subsequent views. It also loads all of the required Javascript libraries required for displaying the data.

2.5.8 v_highchart.jade

The template for the scatterplot view.

2.5.9 v_histogram.jade

The template for the histogram view.

2.5.10 v_vivagraph.jade

The template for the force directed graph view.
The file *index.jade* in "views" was modified to allow the user to navigate to the newly created views.

[6] http://www.highcharts.com/
[7] https://nodejs.org/en/
[8] https://www.mongodb.org/
[9] http://getbootstrap.com/
[10] https://timeline.knightlab.com/
[11] https://docs.angularjs.org/api/ng/service/$http

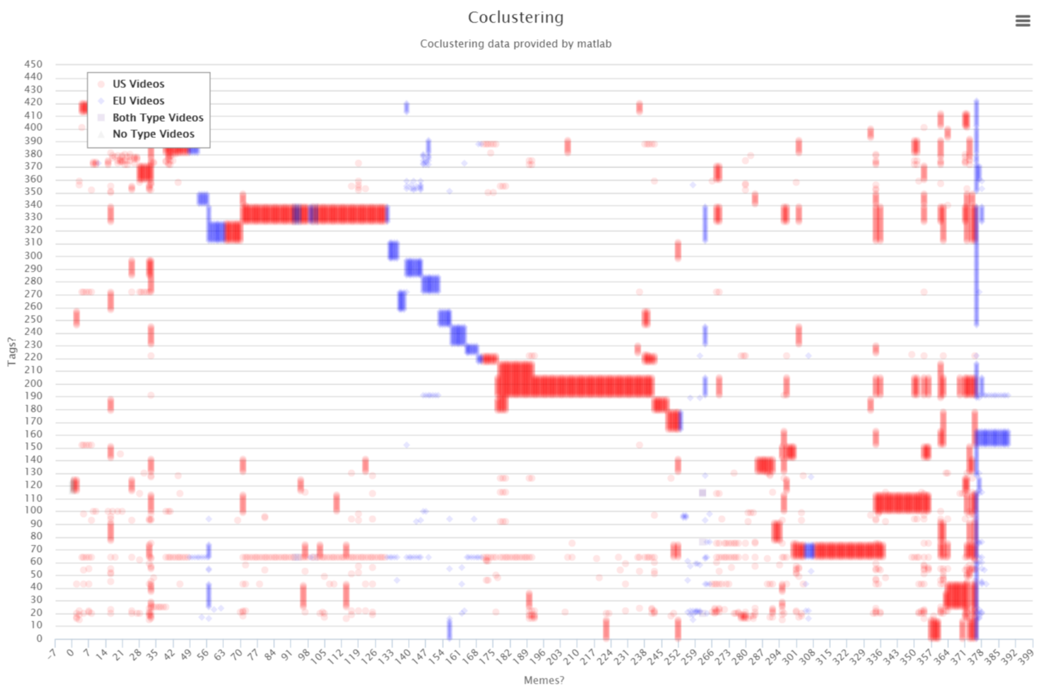


Figure 9: The raw matrix displayed using Highcharts

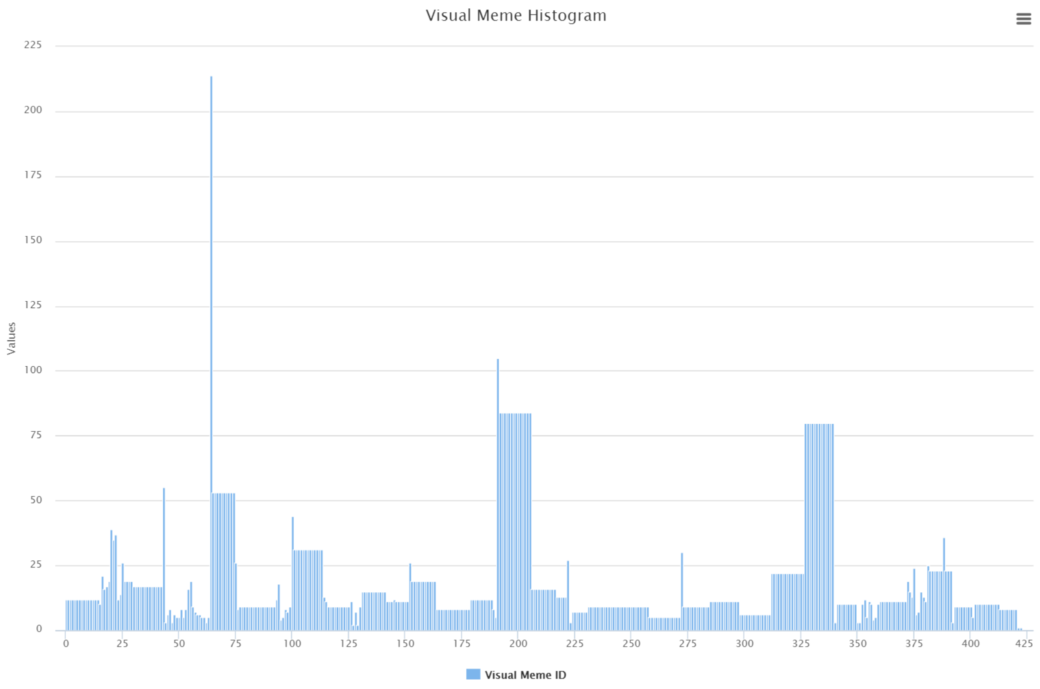


Figure 10: A histogram of the number of tags corresponding to each visual meme

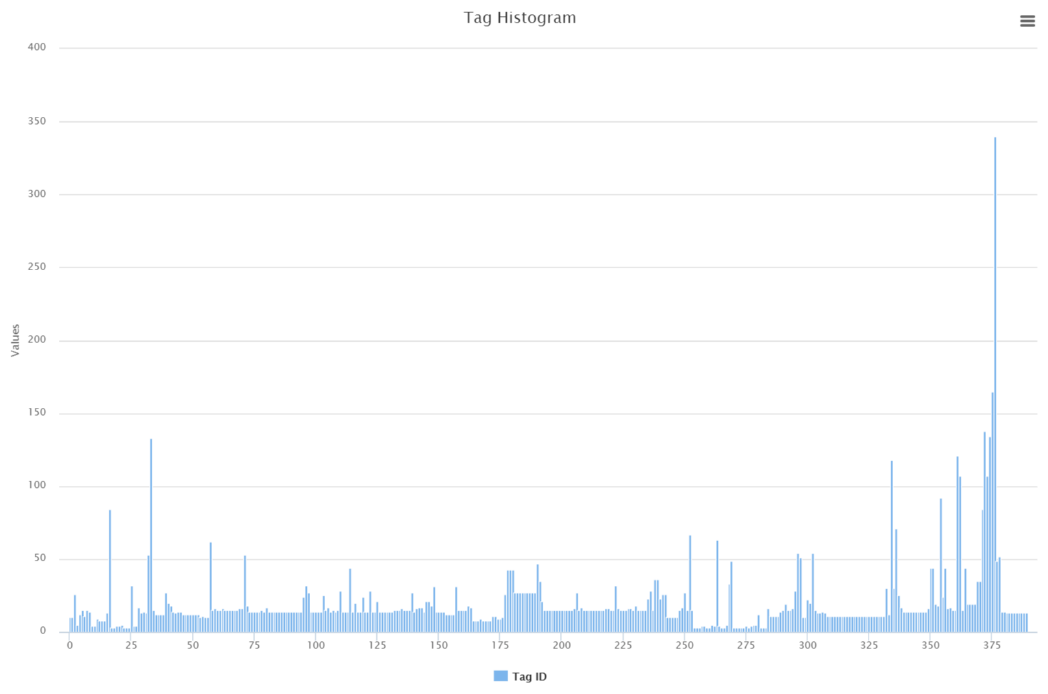


Figure 11: A histogram of the number of the visual memes corresponding to each tag

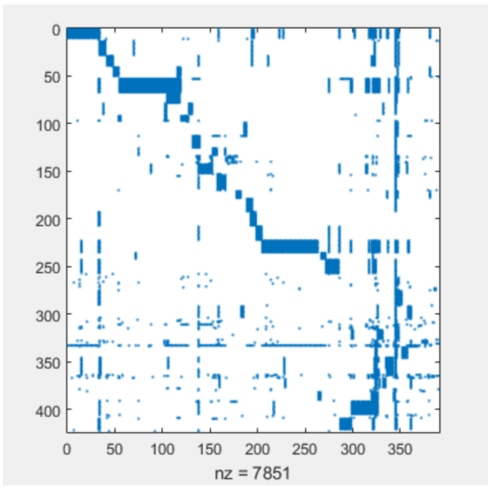


Figure 12: The original matrix generated by Matlab

2.6 Data

In order to get this application working the following data files were generated using the provided Matlab scripts and then stored in the "public/data" directory as the following filenames.

2.6.1 A.txt

During my research I was provided with a series of Matlab files to generate and display a coclustered matrix with visual memes as the y-axis and tags as the x-axis. For the entirety of the project I used one such matrix comparing videos on the Ebola epidemic which featured 390 tags and 423 visual memes. The file itself has each column comma delimited where each value is either a 0 or a 1. Each row is newline delimited.

2.6.2 Cluster1.txt

The Cluster1 file corresponds to a series of clusters of videos that contain a series of visual memes. I was able to use this file to determine whether a particular visual meme corresponded to a US Video or a European video. The format for a particular cluster was filtered using a specific regex that followed the following format:

```
Cluster([0-9]+):
([0-9]+)(../(US_videos|EU_videos)[.]*
```

2.6.3 visual_meme_index

A file that listed all of the visual meme indexes and the cluster that each one belonged to. Allowed me to connect which visual meme belonged to which culture based solely on the cluster.

3. LOGISTICS

3.1 Timeline

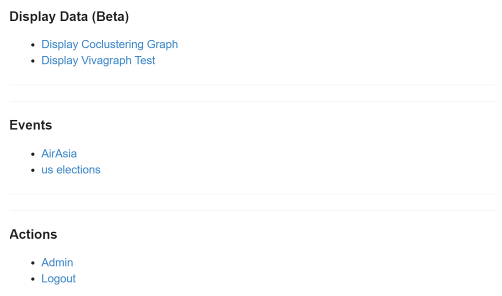
The following timeline best captures the goals of each week and how they were completed. Meetings were held with Professor John Kender and Chun-Yu Tsai weekly every Monday starting on September 28th until the final meeting on December 14th.

September 28th	Read previous report submissions including the NSF proposal, the ICMR papers and the ACM background short paper.
October 5th	Sketch designs for visualizing data and create some demos of different ways of visualizing the matrix provided.
October 12th	Generate rudimentary visualization of memes and tags and use modified alpha values to demonstrate overlap of visual memes and tags in a scatterplot.
October 19th	Implement provided visualization using Bootstrap framework and add force directed layout example.
October 26th	Meeting postponed due to scheduling conflict.
November 2nd	Iterate on force directed layout by adding colors corresponding to videos that originated from the US and videos that originated from Europe. Also add these colors to the scatterplot representation.
November 9th	Find appropriate research in applying block diagonalization to an $m \times n$ matrix.
November 16th	Apply speedup techniques by using arrays instead of lists in the javascript code. Take the rowsum and column sum of visual memes and tags in the matrix and remove the most/least popular. Show strength of relationships in force directed layout with thicker lines.
November 21st	Find research on removing outliers from matrices.
November 30th	Clean up code source and migrate to angular.js to make adding features and maintaining code easier.
December 7th	Implement Reverse Cuthill McKee algorithm and observe block diagonalization effects on the scatterplot.
December 14th	Write report describing process through research.

3.2 Git History

The following is a log of all git commits made over the course of the semester. Git commits were made only when large features were added.

```
commit 25f95a47464da0d79e7f0de92b98397815677dac
Author: David Watkins <djw2146@columbia.edu>
Date:   Fri Dec 11 23:38:09 2015 -0500

Implemented RCM filtering

commit e44a19eb711f552bd42535ce3e3888856a32dac5
Author: David Watkins <djrival7@gmail.com>
Date:   Mon Dec 7 13:35:28 2015 -0500

Added angular js implementation to improve overall
↪  code

commit b984abe297bfb93e954275fb683acf514b7df090
Author: David Watkins <djw2146@columbia.edu>
Date:   Mon Nov 23 08:24:52 2015 -0500

Added additional filtering methods

commit 72e5398eca8177a97c7857e63ead9111f551ed45
Author: David Watkins <djw2146@columbia.edu>
Date:   Mon Nov 16 04:20:24 2015 -0500

Another viva test

commit cbb0ef66fddef03d56d2a99ea85b5b4db100babc
Author: David Watkins <djw2146@columbia.edu>
Date:   Mon Nov 16 03:32:58 2015 -0500
```


Display Data (Beta)
- Display Coclustering Graph
- Display Vivagraph Test

Events
- AirAsia
- us elections

Actions
- Admin
- Logout

Figure 13: Index of the web application

```
Added initial vivagraph.js test, refactored code

commit d4d9378a69332a108adda1e571d1d45dfb450fcc
Author: David Watkins <djw2146@columbia.edu>
Date:   Mon Nov 9 06:16:38 2015 -0500

Initially adding all extra features
```

4. TUTORIAL

4.1 Prerequisites

In order to run the application you must have a copy of *Node.js* and *MongoDB* installed on your system. A copy of the *dump/* directory is also required which has been kept within the compute cluster. Download this file and initialize the database with the command

```
>mongorestore dump/
```

4.2 Getting Started

To run the application first make sure you have an instance of mongod running by running the command

```
>mongod
```

If you have never installed Node.js before, run the following command

```
>npm install
```

in the directory of the cloned repository. Once the dependencies are installed run

```
>node app
```

which will run the web server.

4.3 Viewing the Coclustered Matrix

Once the app is running open an instance of a web browser and navigate to *localhost:3000*. This is the default port, if you have changed the configuration then correct the port and go to that address. Because login credentials have been disabled in my repository, logging in will not be necessary.

You should then click on *Display Coclustering Graph* which will take you to *http://localhost:3000/display*.

The navigation bar on the left of the application contains hyperlinks to the three possible data visualizations of the tool. Both the force directed layout and the highcharts layout feature several parameters to modify the overall view of graph. The histogram view does not have any parameters. The sidebar menu can be closed by pressing the hamburger menu icon on the top-left. The search and triple dot icons in the top-right do not do anything at the moment.

For the highcharts view,

- The alpha value corresponds to the alpha color of each of the dots in the scatterplot

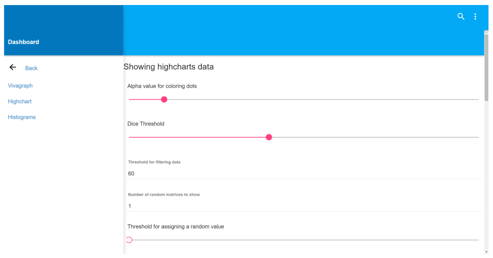
Figure 14: AMD default ui

Figure 15: Initial screen when viewing histograms

- The dice threshold is on a scale of 0.1 to 1 of how similar each column needs to be before it is removed
- The threshold for filtering data is the cutoff for the histogram filter
- The number of random matrices to show is currently not working and neither is generating fake data, so do not use these
- The radius for the dots on the graph controls how big each value on the scatterplot appears
- The filter types correspond to either the histogram filter or the dice filter for reducing the data set in the scatterplot
- The diagonal types correspond to the visual meme, tag, or both being block diagonalized using the RCM algorithm
- The reload button refreshes the scatterplot using the newly assigned parameters.

The force directed layout is largely the same as the scatterplot view,

- The dice threshold is on a scale of 0.1 to 1 of how similar each column needs to be before it is removed



Showing highcharts data

Alpha value for coloring dots

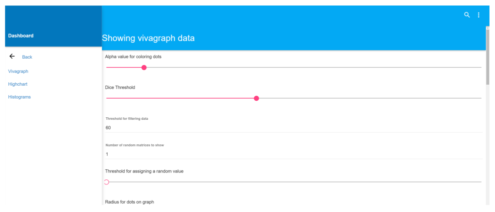
Dice Threshold

Threshold for filtering data
60

Number of random matrices to show
1

Threshold for assigning a random value

Figure 16: Initial view when viewing scatterplot view



Showing vivagraph data

Alpha value for coloring dots

Dice Threshold

Threshold for filtering data
60

Number of random matrices to show
1

Threshold for assigning a random value

Radius for dots on graph

Figure 17: Initial screen when viewing force directed layout

- The threshold for filtering data is the cutoff for the histogram filter
- The number of random matrices to show is currently not working and neither is generating fake data, so do not use these
- The radius for the dots on the graph controls how big each value on the scatterplot appears however this should be removed
- The filter types correspond to either the histogram filter or the dice filter for reducing the data set in the scatterplot. The dice filter should not be used with the force directed layout
- The graph types correspond to which axes of the matrix correspond to the nodes in the graph. The options are making visual meme nodes, making tags nodes, and making both nodes.
- The reload button refreshes the forced directed layout using the newly assigned parameters.

The colors for all of these views are hardcoded so that US videos are red, European videos are blue, videos that are clustered to both are purple, and videos that belong to neither are gray.

4.4 Code Listing

All code mentioned in this report can be found and downloaded from here (https://github.com/DavidWatkins/Cross-Cultural-Video-Aggregator)

5. RESULTS AND ANALYSIS

5.1 Analysis

5.1.1 *Forced Directed Layout Representation*

Originally while using Vivagraph to simulate the force directed layout of the graph was very slow and caused development to take much longer than with d3. Once d3 was used the iterations between views of the force directed layout were faster. Adding additional features in d3 was very simple and the community support for the framework made designing the app even easier.

As for the data visualization itself, it was almost impossible to draw the graph when all of the nodes were generated. The browser I was using was not able to process the immense amount of data required to draw the graph. I had to consistently use the histogram filter with a cutoff of 60 in order to get a performant representation of the relationships between the nodes. When using the visual memes as nodes, there were clear clusters being formed between several visual memes and their tag links. The only novel view was where both the visual memes and the tags were nodes, which allowed for an interesting grouping of nodes being interconnected through a series of tags and visual memes.

More analysis needs to be conducted as to which visual memes each of the groupings correspond to and how each of these clusters arise within the data itself. This will require building additional functionality to view the visual meme frames as well as the tags within the data visualization UI.

5.1.2 Scatterplot Representation

In my testing of different representations of the data using a scatterplot, I found certain settings were best when combined with others. One combination I was unable to test due to limitations in how I could code the result was the dice filter with the block diagonalization. I tested block diagonalization of both visual memes, tags, and both at the same time with the histogram filter being applied to the data with a cutoff of 60. I attempted to block diagonalize at smaller values and found the graph was not as diagonalized. The algorithm used to generate the diagonalization has one bug in it causing the matrix to be flipped on the x-axis, something I did not have time to fix. Some further analysis that should be done is how linearly related the diagonalized data is relative to the disorganized version.

Using the dice coefficient showed that summarizing the columns greatly reduced the number of tags in the data. This did, however, not integrate well with other features in the application (RCM and force directed layout). Because of this the dice filtering should only be used to visualize a summary of the data.

I also added a small setting allowing the user to adjust the alpha value of the points on the graph so as to make them more visible. After some simple testing, an alpha value of 0.2 was the most convenient for viewing the data, but because this was an aesthetic choice I have the option for changing the alpha value in the UI.

5.2 Next Steps

The following is a series of next steps that need to be taken with respect to development of this application

- The data is currently being loaded from a series of text files. These files need to be migrated to a MongoDB database as well as create a utility for adding additional data.
- The RCM algorithm only allows for one axis to be diagonalized at a time due to its requirement of an $n \times n$ matrix. Additional block diagonalization algorithms should be explored that allow for $m \times n$ matrices.
- The login functionality that existed in the original application was disabled for the development of the data visualization. This needs to be integrated and re-enabled so the final consumer application can be created. This also means that the UI needs to be standardized across the data visualization and the consumer application which will require migrating the data visualization to bootstrap or the pre-existing codebase to Angular.
- The force directed layout and scatterplot layout lack visual information regarding which visual memes and tags the indexes correspond to. The user of the data visualization should be able to mouseover the matrix to view the corresponding visual memes and tags.
- The RCM implementation currently needs to be flipped across the x-axis so that it is properly diagonalized top-left to bottom-right as opposed to bottom-left to top-right.
- The dice filtering algorithm is not currently compatible with either the block diagonalization or the force directed layout. The codebase should be updated to support the additional functionality.
- One of the original goals of the research project was to support a timeline of matrices and view them conveniently. This should be reincorporated using the Timeline.js that exists in Skylar's application.

6. CONCLUSION

Throughout the semester I have worked on developing a reliable, easily customizable and visually appealing user interface using a variety of Javascript frameworks so as to make the data visualization of visual memes and tags in Chun-Yu Tsai's research more accessible. I have succeeded in creating a proper visualization of the data in scatterplot, force directed layout, and histogram representations. Moving forward there are a variety of paths that this research project can take that all require expertise in Javascript and good cluster visualization techniques.

7. REFERENCES

[1] Science and samgaetang.blogspot.com. Science & samgaetang: Reverse cuthill-mckee ordering in python & cython, 2016.

[2] E. Cuthill and J. McKee˙ Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM.

[3] Spencer Patty. Compressed row storage (crs) format for sparse matrices, 2014-4-2.

[4] People.revoledu.com. JaccardâĂŹs coefficient, 2016.

[5] Chun-Yu Tsai and John R. Kender. Tracking cultural differences in news video creation. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, pages 951–954, New York, NY, USA, 2015. ACM.

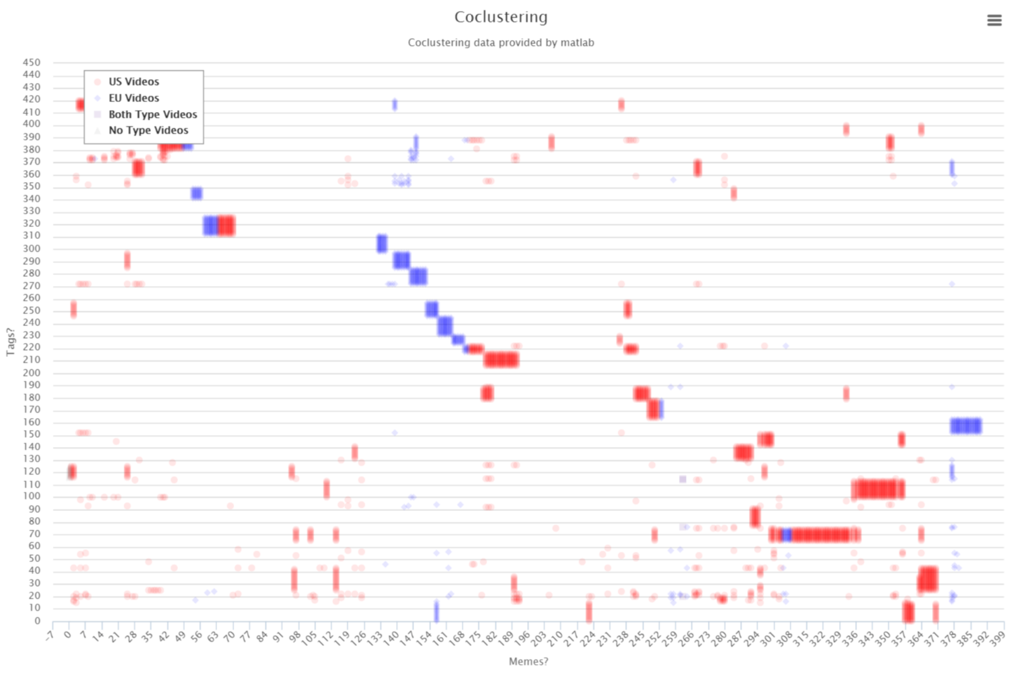


Figure 18: Histogram filtered matrix with a cutoff of 20. Not clean enough to apply block diagonalization

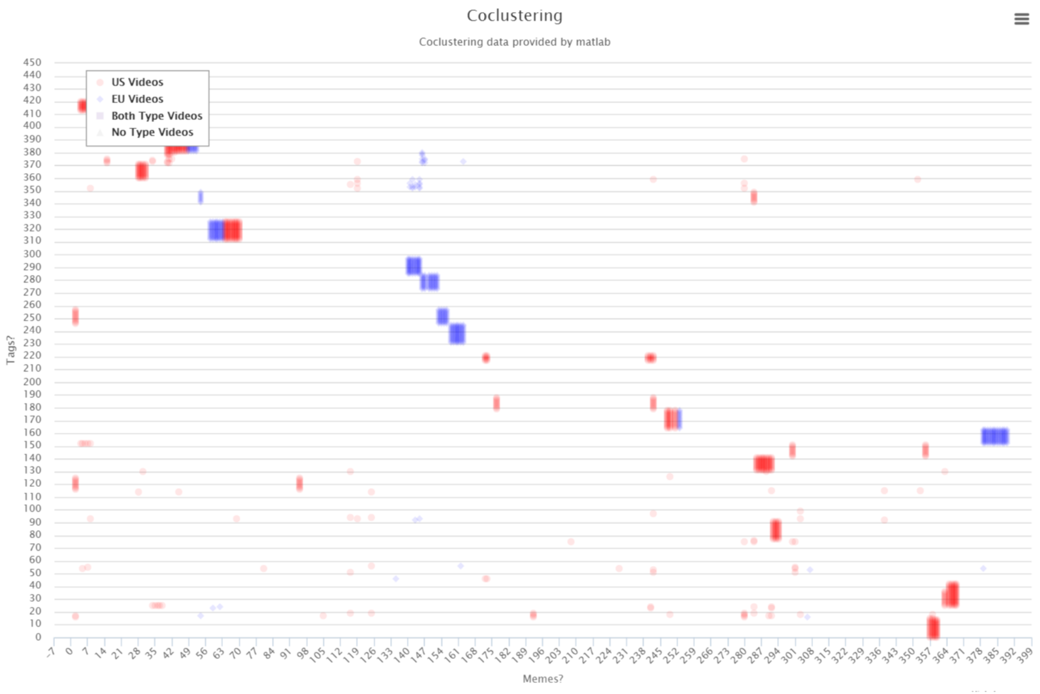


Figure 19: Histogram filtered matrix with a cutoff of 60. Sparse enough to run the RCM algorithm on.

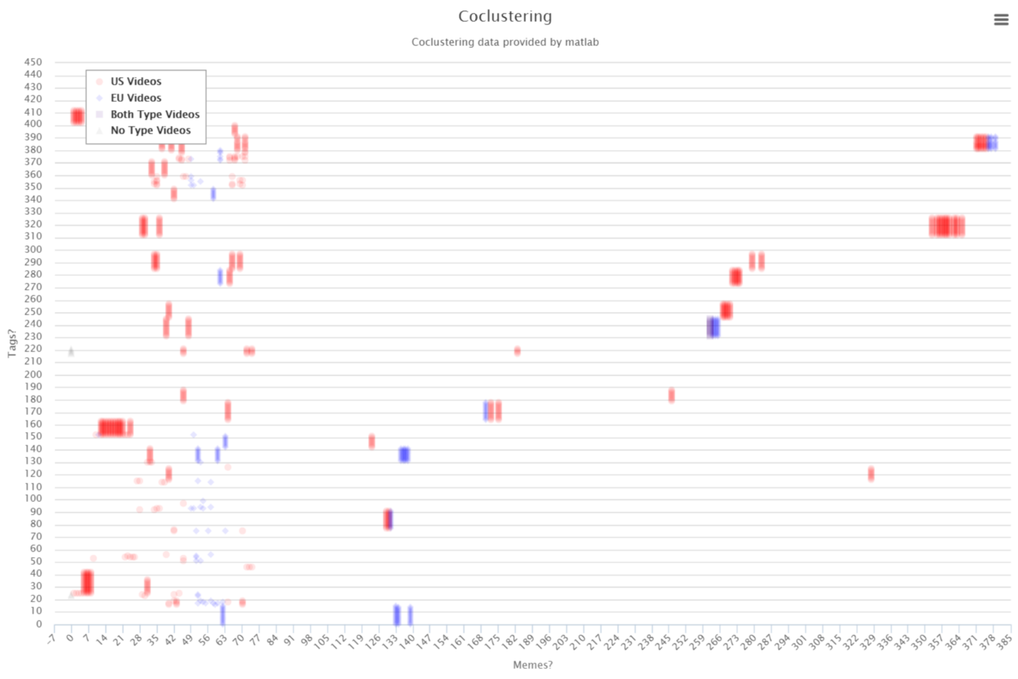


Figure 20: Histogram filtered matrix with a cutoff of 60 and block diagonalized using the tags. Proved to be an effective diagonalization with the larger columns at the bottom-most left-most position.

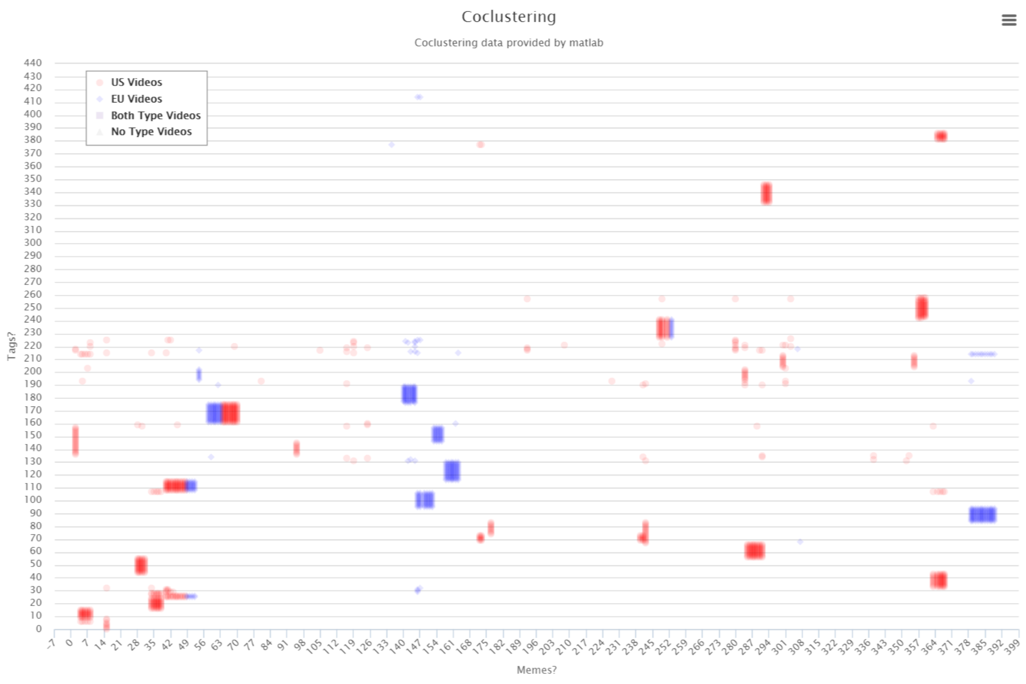


Figure 21: Histogram filtered matrix with a cutoff of 60 and block diagonalized using the visual memes. Not as organized as the previous tag diagonalization.

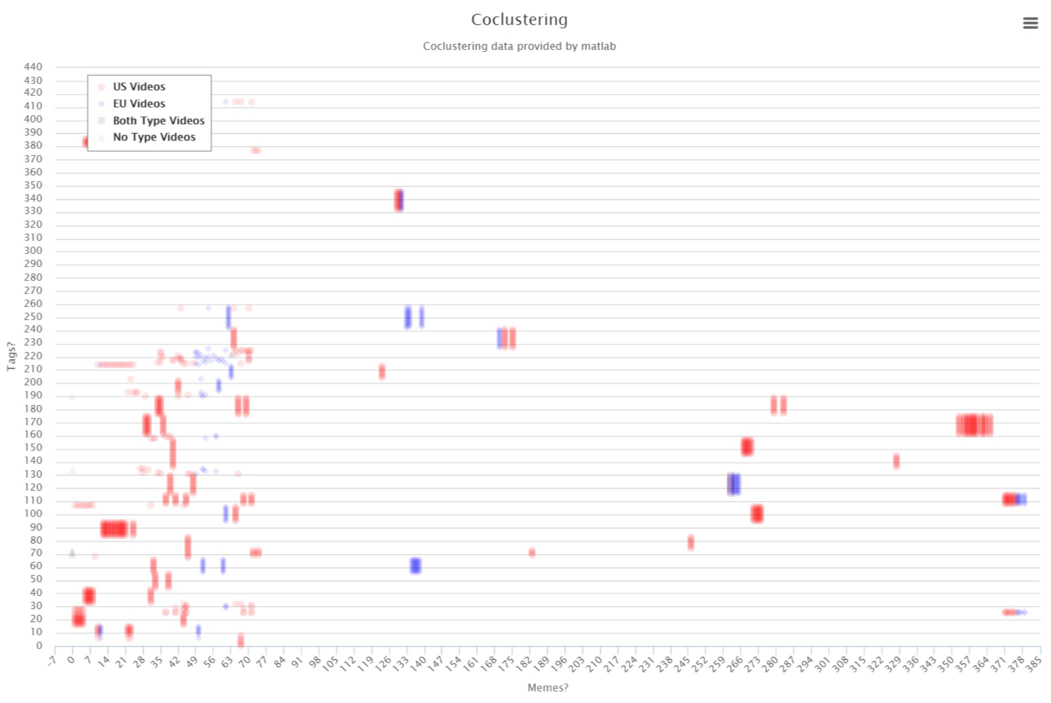


Figure 22: Histogram filtered matrix with a cutoff of 60 and block diagonalized using the visual memes and tags. Because the block diagonalization for both axes occur independent of one another the result is disorganized.

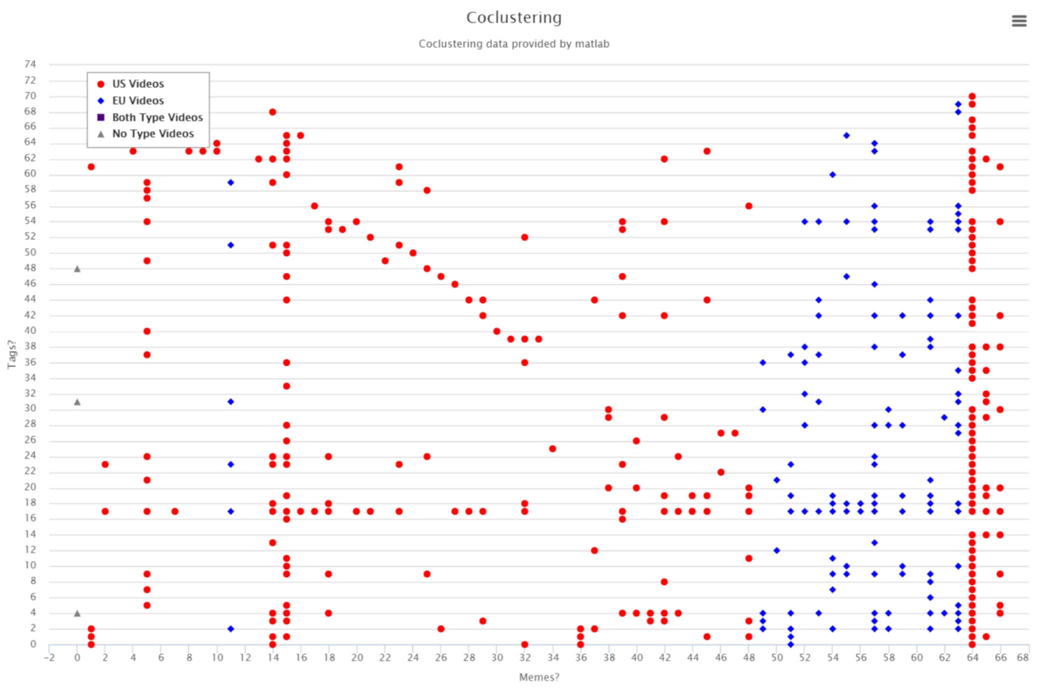


Figure 23: Applying the dice filter to the matrix with a dice coefficient cutoff of 50% similarity

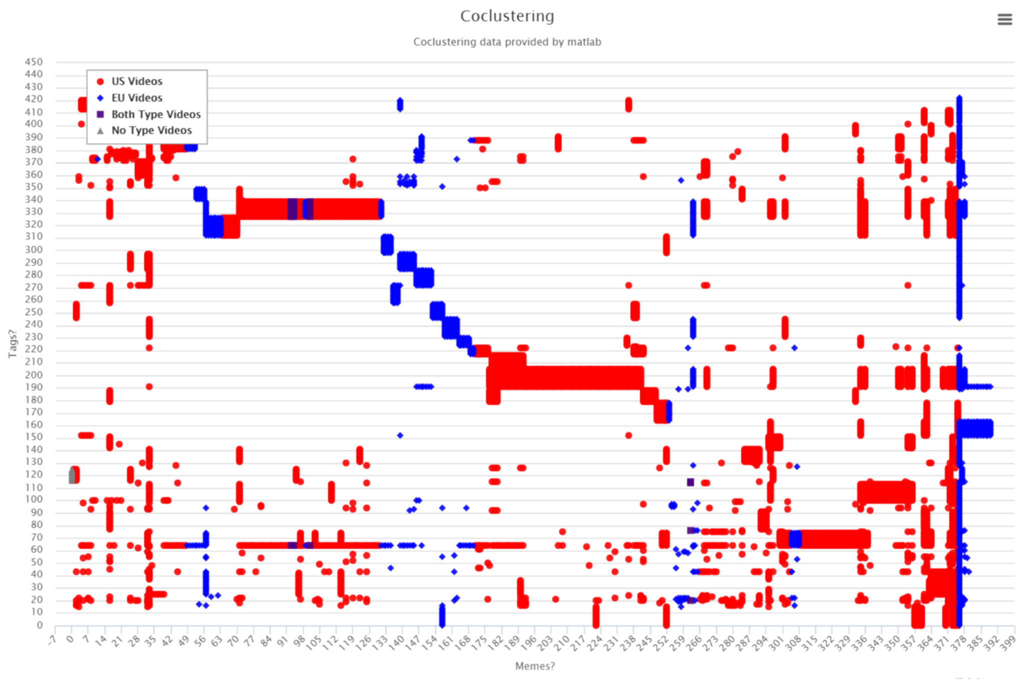


Figure 24: Showing a higher alpha value for each position in the plot