

# Topical Clustering, Summarization, and Visualization

Dan Siroker  
dsiroker@stanford.edu

Steve Miller  
millersj@stanford.edu

## Abstract

The growing collection of academic papers available on the web provides a terrific resource for researchers. With this volume comes a need for organization, a task which CiteSeer and other sites have attempted to tackle. We set out to extend the functionality provided by CiteSeer through an investigation of clustering by topic similarity, experimenting with visualization and summary of clustering results. We had three main goals in mind. First, we hoped to develop a sensible clustering of the articles based on textual similarity. In doing so, we ended up using CiteSeer's similarity results as partial input to our clusterer. Second, we hoped to provide a meaningful visualization of some of the topics represented in the article corpus. Finally, we wanted to investigate the possibility of automatically naming each cluster given the features present in each article and the article's similarity to the closest centroid. With such a large pool of articles, we felt that automatic generation would be preferable to a hand-crafted taxonomy.

# Contents

## 1. Introduction

- 1.1 Background
- 1.2 Related Work

## 2. Raw Data Gathering

- 2.1 Newsgroups
- 2.2 Scientific Articles
  - 2.2.1 Crawling
  - 2.2.2 Document Features

## 3. Clustering

- 3.1 Algorithms and Approach
  - 3.1.1 K-means clustering
  - 3.1.2 Feature weighting
  - 3.1.3 Feature selection
  - 3.1.4 Vector similarity
- 3.2 Implementation Notes
  - 3.2.1 Time concerns
  - 3.2.2 Memory concerns
- 3.3 Results and Performance
  - 3.3.1 Newsgroup results
  - 3.3.2 Article results

## 4. Visualization

- 4.1 Algorithms and Approach
  - 4.1.1 Multidimensional Scaling
  - 4.1.2 Generating Similarity Graphs
- 4.2 Results and Performance
  - 4.2.1 Article Graphs
  - 4.2.2 Cluster Graphs

## 5. Cluster Summarization

- 5.1 Algorithms and Approach
  - 5.1.1 Introduction
  - 5.1.2 Maximum Vector Difference
- 5.2 Results and Performance

## 6. Conclusions

## 7. References

# **1. Introduction**

## **1.1 Background**

Clustering has become an increasingly important topic with the explosion of information available via the web. Several groups have investigated webpage clustering, which often goes beyond comparisons of textual similarity to include and focus on link analysis.

The CiteSeer online service represents a comprehensive attempt to collect and organize scientific literature on the web. This served as a valuable resource in our hunt for data, as well as an interesting model for study. Lawrence, Bollacker, and Giles provide a good overview of the methods and goals of the project. The CiteSeer approach to similarity seems focused on citation graph analysis, which produces reasonable results. There is also a section of “similar articles” provided for each article. We hoped to take some of this work as a starting point and apply textual clustering techniques to refine the similarity relationships given by CiteSeer, plug the results into a more meaningful visualization tool, and automatically generate sensible labels for the clusters obtained.

## **1.2 Related Work**

The topic of cluster naming or summarization is closely related to the goal of building searchable subject hierarchies. Nevill-Manning, Witten, and Paynter took a different approach to such a goal, using grammar induction to infer a hierarchical structure in a corpus. We hope to achieve similar success within a clustering framework, re-using some of the same methods employed to group pages as a means to give names to clusters. In addition, a graphical visualization of clusters and their proximity should provide a more informative interface to the underlying articles.

# **2. Raw Data Gathering**

## **2.1 Newsgroups**

For each article referenced on CiteSeer, there is only a limited amount of text available without downloading a full-text article. A portion of the abstract is made available, as well as some information regarding articles deemed related by CiteSeer. With only 50-100 words available in each abstract, we decided that our clusterer should be developed and tested on

a more substantial data set. For this, we turned to a collection of newsgroup postings available in Stanford University's Leland space at

```
/afs/ir/data/linguistic-data/TextCat/20Newsgroups/
```

This folder contains two sets of newsgroup postings, each of which is a directory containing several collections of postings for different newsgroups. The topics range from computer graphics to religion. Because there are roughly one thousand files per newsgroup, each of which is considerably larger than the abstract snippets on CiteSeer (on the order of a few thousand words), these postings provided us with a suitable data set for developing and evaluating our clusterer. In addition, the separation of postings into different folders affords a built-in metric for evaluation, allowing for calculation of external similarity.

## **2.2 Scientific Articles**

### **2.2.1 Crawling CiteSeer**

In order to generate a corpus of scientific articles, we wrote and ran a web crawler on CiteSeer's website. Our eventual goal was to download the full body text for each article and use the similarity of the article's text as our criteria for clustering. We realized early on that this would be infeasible if we wanted to have a reasonably large-sized corpus since each article would need to be retrieved from the internet, stored locally, and converted to plain text from PDF or postscript. Instead, we used the link structure of the CiteSeer website to dictate what our features would be for each article. Since CiteSeer actually already tries to determine similar or related documents for every document in its repository, we used this to our advantage.

Given a document  $d$  in CiteSeer's repository there is a webpage  $p$  that contains meta-information about document  $d$ . This webpage  $p$  has the document's title, date of publication, authors, portion of the abstract, a link to the actual document  $d$  and links to similar or related documents. Let us call this set of similar or related documents  $S$ . Since CiteSeer provides a link from page  $p$  to every other page in  $S$  we designed our web crawler to follow this link structure. When we ran our crawler we downloaded 200,000 pages totaling 4GB. We didn't actually download the underlying document  $d$ , but instead downloaded every corresponding page  $p$ .

## 2.2.2 Document Features

In order to run standard clustering algorithms over our set of documents we had to choose what kind of features would serve to represent the document in our model. For most text clustering applications others have used a vector  $v$  to represent a document  $d$  by assigning a new dimension for every unique word type in  $d$ . The values of these dimensions are determined by feature weighting (which is discussed in section 3.1.2).

The screenshot displays a CiteSeer page for the document "Digital Libraries and Autonomous Citation Indexing (1999)". The page includes a header with the title, authors (Steve Lawrence, C. Lee Giles, Kurt Bollacker), and the source (IEEE Computer). It features navigation links such as "Home/Search", "Context", and "Related". A "View or download:" section provides links to the document in various formats (PS.gz, PS, PDF, DjVu, Image, Update, Help). The page also includes a "Rate this article:" section with a star rating and a "View Comments" link. The abstract discusses the World Wide Web's impact on scientific information access and introduces the creation of digital libraries. Below the abstract, there are sections for "Context of citations to this paper:", "Cited by:", "Active bibliography (related documents):", "Similar documents based on text:", and "Related documents from co-citation:". Each section contains a list of related documents with their titles, authors, and years, along with a "Correct" link for each.

**Digital Libraries and Autonomous Citation Indexing (1999)** (Make Corrections) (46 citations)  
Steve Lawrence, C. Lee Giles, Kurt Bollacker  
IEEE Computer

View or download:  
[nec.com/homepages/...ute98letter.ps.Z](#)  
[nec.com/homepages/...omputer98a4.ps.gz](#)  
[com/homepages/gile...mputer.DLACI.ps.Z](#)  
Cached: [PS.gz](#) [PS](#) [PDF](#) [DjVu](#) [Image](#) [Update](#) [Help](#)

From: [nec.com/homepages/lawren...publ](#) (more)  
From: [nec.com/homepages/lawrenc...latex](#)  
Homepages: [S.Lawrence](#) [C.Giles](#)  
[K.Bollacker](#) [HPSearch](#) (Update Links)

**CiteSeer** Home/Search Context Related  
Scientific Literature Digital Library

Autonomous creation of citation indices and advantages for scientific communication and progress

Rate this article: 1 2 3 4 5 (best)  
[View Comments \(0\)](#)

**Abstract:** The World Wide Web is revolutionizing the way that researchers access scientific information. Articles are increasingly being made available on the homepages of authors or institutions, at journal Web sites, or in online archives. However, scientific information on the Web is largely disorganized. This article introduces the creation of digital libraries incorporating Autonomous Citation Indexing (ACI). ACI autonomously creates citation indices similar to the Science Citation Index R. An ACI...  
(Update)

Context of citations to this paper: [More](#)

.... based on textual similarity are especially useful for databases found on the Web [1] or obtained by extracting information from text [6, 13, 11], where descriptive names generally exist but Permission to make digital or hard copies of all or part of this work for personal or...

...nor an online citation service available. In this case, one may directly extract the reference information from the article s text. CiteSeer [12] and CiteBase have used this approach. Online Reference Service unavailable  
OAI Non OAI Harvester Process Archon Search Engine...

Cited by: [More](#)  
[Low level information extraction - Bayesian Network Based](#) (Correct)  
[Interactive Deduplication using Active Learning - Sunita Sarawagi Sunita](#) (Correct)  
[The widespread adoption of the Web as a mechanism - For Sharing Information](#) (Correct)

Active bibliography (related documents): [More](#) [All](#)  
0.5: [CiteSeer: An Automatic Citation Indexing System - Giles, Bollacker, Lawrence \(1998\)](#) (Correct)  
0.2: [Indexing and Retrieval of Scientific Literature - Lawrence, Bollacker, Giles \(1999\)](#) (Correct)  
0.2: [Determining the Publication Impact of a Digital Library - Kaplan, Nelson \(2000\)](#) (Correct)

Similar documents based on text: [More](#) [All](#)  
2.1: [Autonomous Citation Matching - Lawrence, Giles, Bollacker \(1999\)](#) (Correct)  
1.0: [Searching the Web: General and Scientific Information Access - Lawrence, Giles \(1999\)](#) (Correct)  
0.9: [Distributed Error Correction - Lawrence, Bollacker, Giles \(1999\)](#) (Correct)

Related documents from co-citation: [More](#) [All](#)  
8: [The anatomy of a large-scale hypertextual Web search engine - Brin, Page](#)  
7: [Accessibility of information on the Web \(context\) - Lawrence, Giles - 1999 - http://www.wisconsinmetrics.com/](#)  
7: [Authoritative sources in a hyperlinked environment - Kleinberg - 1997](#)

Figure 1: Screen shot of a typical CiteSeer page: <http://citeseer.nj.nec.com/22863.html>

However, since it was infeasible to use the actual document text of  $d$  as the set of features for the document, we chose to do something slightly different. We combined the document's title with the titles of all the pages in  $S$  that page  $p$  linked to as a long string of words. These words would serve as features for document  $d$ . For example, take the article given as Figure 1. We would generate the following bag of words to build the vector to represent this document:

“Digital Libraries and Autonomous Citation Indexing CiteSeer: An Automatic Citation Indexing System Indexing and Retrieval of Scientific Literature Determining the publication Impact of a Digital Library Autonomous Citation Matching Searching the Web General and Scientific Information Access...”

This proved to be a very useful set of words to use to build our vector because it concisely contained meaningful keywords. Since we ended up looking at unigrams, the order and sentence structure were irrelevant. Appending all the titles of related documents utilized the work that CiteSeer had done while at the same time saving us a lot of work by avoiding having to look at the whole document text. If a particular keyword appeared in several related document titles, its vector weight would be dramatically higher than words that only appeared once. We essentially created a way to combine all the related documents into one useful representation.

Once we have a bag of words for a given document, we create a feature vector to represent the document. The dimensions of the vector represent the word types in the bag and the weight of the dimensions are determined by the weighting scheme described below. In the following section we will show how to cluster a set of these feature vectors and describe in detail how we weight these features.

## **3. Clustering**

### **3.1 Algorithms and Approach**

#### **3.1.1 K-means clustering**

Once we have feature vectors for each article, we have two main tasks to perform.

First, we must select an initial value for each of the  $k$  means. The results of the clustering algorithm can be highly sensitive to these initial

choices, so the initialization method is critical. Based on recommendations in prior work, each mean was initialized as follows:

1. Select a random article from the data set.
2. For each feature in that article's vector of feature scores, randomly perturb the score (using a fractional multiplier), and place this feature score in the mean's vector of feature scores.

Second, we must iteratively assign the articles to the closest cluster, then recalculate the means based on the feature scores of the articles assigned to the cluster. This iteration continues until one of two events occurs. For ensured termination, there is a maximum number of iterations imposed on the algorithm. For optimization, we can halt the iteration as soon as the means haven't changed much. To formalize this concept of equilibrium, we compared each updated mean to its value on the prior iteration using the same similarity measure employed for article-cluster comparison. If the product of these similarities over all means was above a user-defined confidence measure, the clustering process can halt, as not much further movement of the means or article reassignment will occur.

We had to make several choices when designing our algorithm. The first choice we had to make was between hierarchical and flat clustering. Due to the magnitude of the clustering task that we hoped to undertake (several thousand articles), we felt that a hierarchical clustering approach was intractable on account of its time complexity,  $O(n^2)$ . Flat clustering would provide a more efficient approach ( $O(n)$ ), and it could easily be adapted to incorporate new articles through an iterative K-means approach. In addition, since our goal was to find a measure of relatedness among various research topics present in the CiteSeer archives, all we really needed was flat clustering. An automatically generated hierarchy might be helpful for browsing articles by topic, but that was not our goal. Finally, the hierarchical clustering algorithms which we investigated all shared a binary branching structure. While simple, this does not necessarily represent the data best.

One drawback of the flat clustering approach is that most algorithms rely on a predefined number of clusters, which may be arbitrary and could alter the effectiveness of the clusterer. This is an acknowledged challenge that we hoped to counter simply by testing different numbers of clusters for a given data set to see which ones worked out the best.

Our next decision was to choose between hard and soft clustering. We eventually opted for the hard clustering K-means method rather than the soft clustering provided by an adaptation of the EM algorithm. The EM algorithm does provide more information – rather than simply

assigning an article to a cluster, it produces a probability of assignment of a particular article to each cluster. Thus, an article can “belong” to multiple clusters if no membership probability is dominant. Despite these advantages, we felt that the K-means algorithm was better suited to our task. First, the visualization tool that we chose to employ represented clusters as nodes in a graph connected by edges weighted by the similarity of the two clusters. Similarly, articles were connected to their parent cluster via a comparably weighted edge. If we chose soft clustering, the visualization would become more difficult. The weighting on each edge would have to somehow incorporate the probability of that edge being relevant. Second, rather than one edge connecting an article to its parent cluster, we would need  $k$  edges per article. In a sample run of 25,000 articles and 500 clusters, this amounts to an explosion in the number of edges by a factor of 500. This could easily overwhelm our visualization tool.

There were other optimizations presented in various research papers regarding better performing variations of K-means. However, we chose to stick with a simpler implementation of K-means for efficiency purposes. When dealing with several thousand data points, we felt that slightly improving clustering should come second to efficiency. An example of our reasoning comes in the choice not to employ Iterative K-means clustering. Iterative K-means is identical to regular K-means, with the exception that the means are updated after each article is assigned to a cluster, rather than waiting for all articles to be assigned to update every cluster mean. Research suggests that this algorithm helps mitigate the algorithm’s sensitivity to initial mean values, thereby lowering the chance that the algorithm will converge on a sub-optimal local maximum. However, in the example of 25,000 articles and 500 clusters, this would require 500 updates for each pass over the article, as opposed to one update with the basic K-means implementation.

### **3.1.2 Feature weighting**

We investigated several methods of feature weighting during the project. In the end, three methods were seriously considered (TF-IDF, TFC, LTC), and we ended up using LTC weighting of words in documents. Entropy weighting was also investigated, but the computational complexity of that method seemed too great for processing such a large corpus. A brief description of the three most appropriate methods follows, including an assessment of why we chose or rejected each method.

TF-IDF (Term Frequency – Inverse Document Frequency) weighting is the simplest of the three weighting measures. It attempts to score the importance of a particular word in a document by taking the product of two terms: term frequency and inverse document frequency. For term frequency, a high rate of word occurrence in a document ought to reflect relevance and importance to the topic of the document. However, there are some generic words (the, an, this) that occur often but perform poorly as distinguishing features of a document. We attempted to eliminate the effects of some of these words by maintaining a set of stop words that weren't tracked or used as features. However, since we wanted our clusterer to remain topic-independent, we didn't want to encode too many stop words that overfit the clusterer to the data available in scientific papers. This is where inverse document frequency fits in well. Dividing the number of documents by the number of documents in which a word appears gives an idea of how document-specific a word is. Usually the log of this IDF is taken to smooth results. The higher the IDF, the more specific a word is to a small subset of documents. Multiplying this by term frequency, we get a relatively good idea of what words distinguish a topic from others.

There are a few shortcomings of TF-IDF weighting. The most glaring of shortcomings is that it does not account for document length. When comparing documents for similarity, a longer document will have more words and thus likely more features. The cosine similarity measure used to compare feature score vectors alleviates some of this problem through normalization, but we wanted to minimize this potential for skewing results and have the scores themselves independent of document length.

TFC weighting uses TF-IDF weighting in the numerator of the score, but it divides this score by the magnitude of the feature vector for the article in question. This effectively normalizes the scores for each word in each article, eliminating the effects of differences in document length. This is close to the approach that we used, except we used the slight modification of LTC weighting to help reduce the effects of large differences in term frequency.

LTC weighting is almost identical to TFC weighting except that it takes the log of the numerator in an attempt to smooth the weights some. This way, words with extremely high frequency won't dominate as much – it's important for a word to appear many times, but the difference between appearing 50 times and 51 times shouldn't be as great as the difference between appearing no times and one time. We used a variant of this method (Figure 2), where the numerator is identical to the

modified TF-IDF weighting presented on pg 543 of Manning and Schutze.

$$\text{lfc}(i,k): \frac{(1.0 + \log(f_{ik})) * \log\left(\frac{N}{n_i}\right)}{\sqrt{\sum_{j=1}^M \left[(1.0 + \log(f_{jk})) * \log\left(\frac{N}{n_j}\right)\right]^2}}$$

**Figure 2:** lfc weighting for feature i in document k from feature set M

### 3.1.3 Feature selection

When trying to assemble several thousand data points into several hundred clusters, feature selection becomes an important task. We chose not to use every word from every document; if we did, our feature vectors would end up with hundreds of thousands of dimensions. Computationally this was too ambitious for our purposes. So, we needed a way to cut down the number of features used once we had reached the clustering stage. In the end, we settled upon a relatively crude heuristic, selecting the M most frequent words in a particular document and adding them to a global set of features. When it came time to establish feature score vectors for each article, we considered only those words in that global feature set.

Although not too complex, this approach actually has performed very well in our findings and in those of previous experiments in a slightly different form. A related and popular method is to choose a frequency threshold and admit words that occur at least that many times in some document. We essentially have a threshold, albeit one that is relative to the document rather than a fixed length. Ideally, this should also help negate any differences caused by varying document lengths, as shorter documents wouldn't suffer from having fewer words admitted as distinguishing features.

There are other methods of feature selection that are more mathematically complex, most notably Information Gain and Chi-Squared analysis. However, both of those feature selection systems seemed inappropriate for our task. On a high level, a threshold provided us with a quick and dirty heuristic that would scale well to the size of the corpus we aimed to use. In addition, the papers consulted when selecting features did not advocate either method over threshold selection.

### 3.1.4 Vector similarity

Given a vector of feature scores for each article, K-means requires a method for comparing two such vectors. The simple, common, and effective approach that we employed was cosine similarity. This amounts to taking the dot product of the vectors divided by the product of the vector magnitudes (Figure 3). This is a relatively efficient procedure – it is linear in the size of the feature vectors. Because we only store scores for those words which actually appear in an article, the procedure is actually linear in the number of features contained in a particular article, which will often be much smaller than the full feature set itself. All non-existent entries are assumed to be zero.

The principle of cosine similarity extends well to the high-dimensional vectors that come out of the text processing. The similarity measure also accounts for document lengths; division by the product of vector magnitudes will normalize the product. This effectively will determine whether or not two vectors “point” in a similar direction in the high-dimensional space, regardless of the magnitude of the vectors themselves. Another nice property of the cosine similarity measure is that it produces a result between  $-1$  and  $1$ . When evaluating the stability/equilibrium of the K-means iteration process, we can check to see if the sum of the similarities between each cluster mean and its corresponding value on the last iteration is greater than some threshold  $k$ .

$$\text{sim}(v_i, v_j) = \frac{v_i \bullet v_j}{|v_i| * |v_j|}$$

**Figure 3:** Cosine similarity measure

### 3.1.5 Word Stemming

To minimize the effects of inflection and morphological variations of words, we pre-process each word using a provided version of the Porter stemming algorithm. Since words distributions are generally sparse even in a large corpus, the stemming algorithm ought to help gather morphologically related words into a single feature.

## 3.2 Implementation Notes

### 3.2.1 Time Concerns

The article data that we hoped to process was quite large in two respects. First, we aimed to process anywhere between 10,000 and 200,000 articles, gradually increasing the number as our relative successes and

difficulties suggested. Second, when grouping these articles into 100 to 500 clusters, the K-means algorithm can take some time to pass through a single iteration, having to test each article for similarity to all means, then having to recalculate each of those means.

Due to these time concerns, we tried to implement our algorithms with efficiency in mind. We chose to use hashed data structures whenever possible, most notably java HashTable and HashSet classes. Each article stored a hash table mapping words to frequencies, which was later re-used to store word-weighting mappings. This HashTable, once modified to contain word-weighting mappings, was used to implement our feature vectors for the vector similarity calculations. Since we only wanted to store non-zero feature scores for each article (see section 3.2.2), comparison of feature vectors wasn't as simple as comparing parallel elements in two arrays. As a result, we kept HashTables, allowing iteration over one HashTable and lookup in a corresponding table. In our minds, this provided the advantages of space efficiency by storing only non-zero scores, while allowing for quick cosine similarity calculations that leverage the speed of hash lookups.

### **3.2.2 Memory Concerns**

As with time concerns, the sheer size of the data that we aimed to work with mandated space efficiency. However, in some cases, space efficiency was sacrificed for computational ease and speed. For example, we maintained a static table of document frequencies rather than calculating IDFs on the fly so that weighting would be much more efficient. In other places, we tried to use as little memory as possible, pushing class instance variables to the level of local method variables whenever possible. The most significant improvement made was to use the document frequency table as a reference tool for sharing string objects. Since each word frequency table was built based on file input, there could be multiple Strings in different articles that all represented the same text. To solve this duplication issue, we used the document frequency keys as standards for existing String objects. Thus, if a previously unseen word is used, we allocate and store a new String object for it. However, if the word has been seen before, we reused the underlying object referenced in the document frequency table.

## 3.3 Results and Performance

### 3.3.1 Newsgroup Results

The newsgroups were used primarily to train the clusterer and evaluate its performance. The newsgroups came in defined categories, which allowed for external similarity comparison using the fmeasure. We calculated the precision, recall, and fmeasure for each cluster, then averaged the fmeasure over all clusters. Some sample results from clustering with  $k=2$  follow in Figure 4:

Group 1	Group 2	Run 1	Run 2	Run 3	Average
alt.atheism	sci.electronics	.9349	.9817	.9895	.9687
comp.graphics	rec.sport.hockey	.9817	.9817	.8099	.9244
sci.med	sci.space	.8766	.9074	.8686	.8842
alt.atheism	soc.religion.christian	.8480	.8206	.8483	.8390
rec.autos	rec.motorcycles	.8423	.7745	.6525	.7564
pc.hardware	mac.hardware	.5743	.7091	.6203	.6346

**Figure 4:** Average f-measure scores for binary clustering of newsgroup postings

Overall, we were pleased with these results, as they show that in most cases the clusterer was able to properly separate articles. We make two observations regarding this data. First, the clusterer does significantly better when judging between unrelated categories; it has more trouble when distinguishing between two related topics. This is to be expected given the purely statistical approach taken. Second, we see that local maxima can still be a problem. In `rec.autos` vs. `rec.motorcycles`, for example, we see that the third run is significantly worse than the first. This suggests that further investigation of centroid initialization heuristics would be worthwhile in future work.

### 3.3.2 Article Results

The success of the clustering algorithm is much harder to judge as there are no pre-defined categories against which the articles can be compared. We did try to take a look at the article clusters generated and evaluate them in a cruder sense based on our own judgments. Please see the visualization and summarization sections for some examples of our results – the relation of article titles to the generated cluster topics should be a good indicator of clustering success.

## **4. Visualization**

### **4.1 Algorithms and Approach**

#### **4.1.1 Multidimensional Scaling**

One of our goals was to create a tool to visualize the clusters that we generated. In order to graphically visualize the feature vectors for each document we would have to somehow reduce the multidimensional vector onto a two or three dimensional space so that it can be graphically represented.

The idea of multidimensional scaling is straightforward when it is thought of as an optimization problem. Once we have generated a set of feature vectors for all the documents in our dataset we can use the cosine similarity function to generate a similarity matrix for all of the vectors with each other. This similarity matrix could then be converted to a distance matrix where any two vectors with a high cosine similarity would be given a very small distance away from one another and documents with a very low cosine similarity would have a large distance. If we then plot a set of initially randomly placed points on a two-dimensional space we can optimize the Euclidean distance between any two points to be closest to the corresponding ideal distance from the distance matrix. If we can generate a plot of points where the distances between any two points is as close as possible to the corresponding row and column in the distance matrix we have created an appropriate visualization of the multidimensional space.

#### **4.1.2 Generating Similarity Graphs**

In order to implement a program to generate similarity graphs we modified a freely available program used to graph nodes and distances between nodes. We changed the program to accept as input a set of nodes where each node represented a particular article or a particular cluster centroid mean. We then assigned a particular ideal value for the length of edges between each node to correspond to the distance matrix generated by the process described above. The program begins by placing all the nodes at the center of the window and then would iteratively adjust the node positions to reduce the overall stress of the graph (where stress is defined as the average edge length deviation from the ideal distance). The program eventually optimizes the edge lengths such that the nodes remain static and their corresponding location on a

two-dimensional plane represents their cosine similarity. We found that nodes of documents that were very similar were graphed closer to one another than to nodes of dissimilar documents.

We didn't just generate a graph for nodes for each article – we also created a node for the cluster mean of a set of articles. The cluster mean is a vector that represents the sum of all of the features for the articles of a particular cluster.

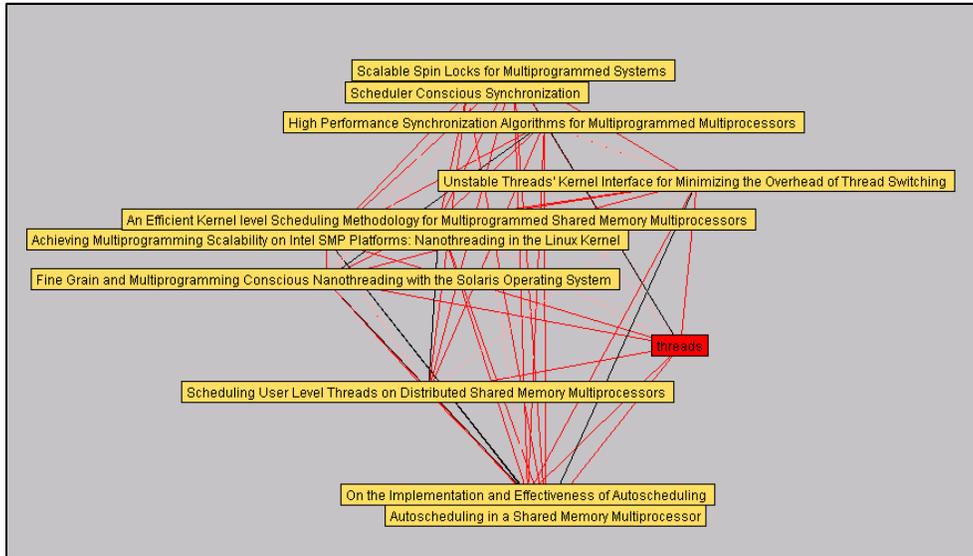
A common problem when we generated the graphs was the difficulty of seeing the names for the nodes if there was a huge grouping in a small area. To solve this problem, the visualization program allows the user to drag and drop particular nodes to try and see the names for nodes that are covered up. It will interactively refit the graph to optimize the distance matrix as you drag the node you want to examine. Another feature of the visualization program is node repulsion. Regardless of the similarity of any two nodes they will also repel each other very slightly so that no two nodes ever occupy the same location. This helps avoid cluttering small areas with many nodes.

## **4.2 Results and Performance**

### **4.2.1 Article Graphs**

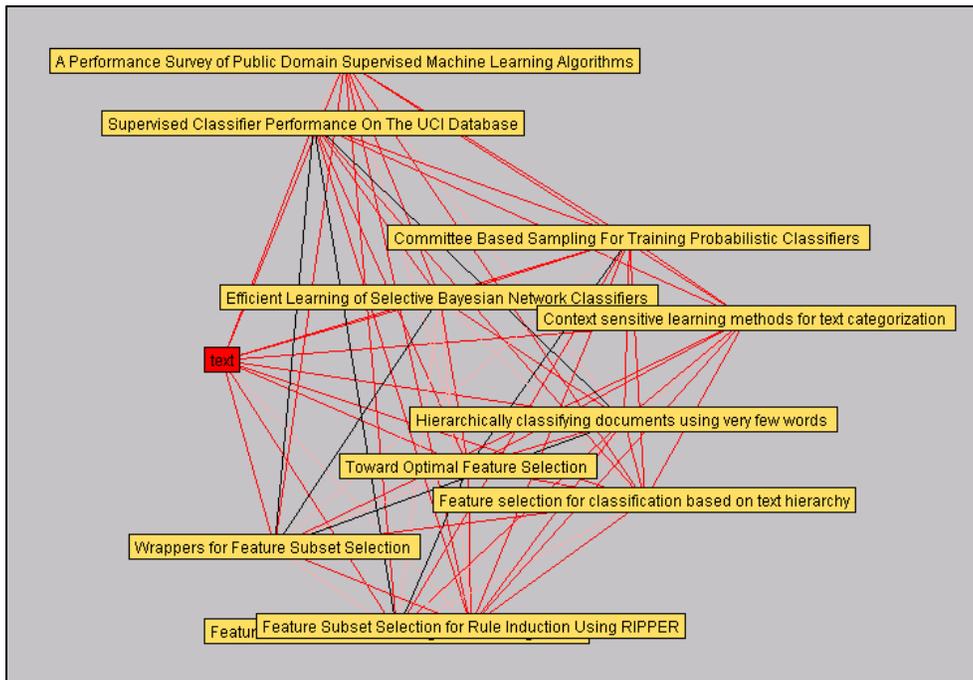
We separated the graphs generated by our visualization program into two categories. First, we graphed all the articles for a particular cluster with their corresponding cluster mean. We did this for every cluster generated. We then created a graph of all the cluster means (without the actual articles). Below is a sample of particularly good visualizations.

Figure 5 shows a graph of all the articles that were assigned to the particular cluster which was labeled “threads”. We will discuss how the word “threads” was chosen in section 5. The other nodes on the graph are labeled by the corresponding document's title. As you can see by this graph similar documents are grouped together: scheduling articles are grouped at the bottom while synchronization and switching are grouped near the top. Another interesting observation is that not every article explicitly has the word “threads” in it. This goes to show that the feature vectors we chose to represent the articles contained enough information in the titles of related documents to make “threads” a prominent feature in the vector.



**Figure 5:** This is a graph of the articles in the cluster labeled “threads”

Figure 6 shows a cluster of documents that are all related to text categorization. It is interesting to note that there were many articles that were unrelated to text categorization that had the word “text” as a feature yet they were not clustered with these articles because they were not similar enough. In this cluster, articles that dealt with “features” were at the bottom and machine learning articles were near the middle and top.

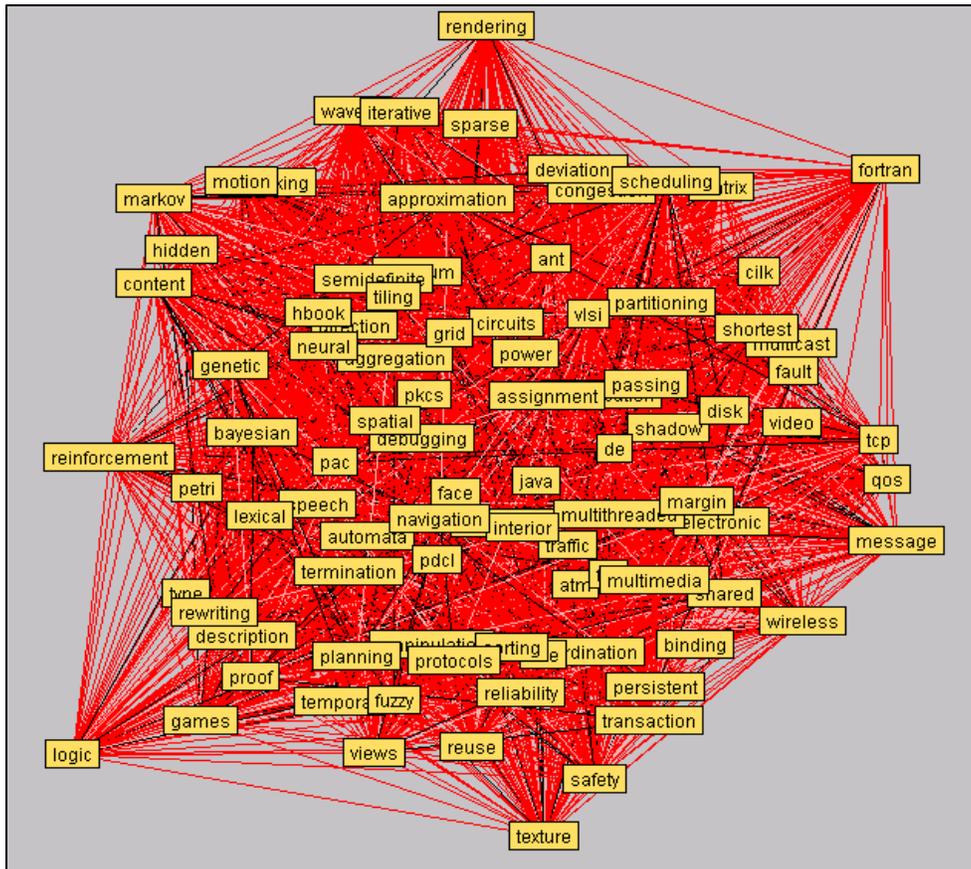


**Figure 6:** This is a graph of the articles in the cluster labeled “text”

## 4.2.2 Cluster Graphs

Graphs of the cluster means proved to be a very good way to visualizing the similarity of groupings of articles. As we will discuss in the next section, we were able to label each cluster with an appropriate word to summarize the contents of that cluster.

The last graph we have included is a graph of clusters and their similarities to one another. We chose to exclude the articles because they would create too much congestion and prevent the reader from seeing the groupings of clusters. Keep in mind that this graph uses only a small portion of all the articles indexed on CiteSeer. Given that the articles to be clustered were related to the clusters given, the graph below does an adequate job of representing their similarity. As you can see, “markov”, “hidden”, and “neural” are all bunched together in the upper left hand corner while “fortran” is all by itself in the upper right. This just shows that articles that deal with fortran aren’t very related to most of the other articles. “java” on the other hand, is right in the middle of the graph.



**Figure 7:** This is a graph of 100 clusters generated from 10,000 articles.

## 5. Cluster Summarization

### 5.1 Algorithms and Approach

#### 5.1.1 Introduction

In order to come up with a name for nodes that represented cluster means, we developed a simple algorithm to determine the most unique feature of a particular cluster compared with all the other articles. The method we chose can't be empirically evaluated because there is no "correct" answer for how to label a cluster mean. We did, however, choose a method that generated qualitatively feasible results.

#### 5.1.2 Maximum Vector Difference

Given a mean cluster vector  $c$ , we can generate a characteristic summarization name for  $c$  by comparing the components of  $c$  to the components of another cluster mean  $e$  which represents a mean vector of everything else that isn't used to generate  $c$  (we call it  $e$  because it is a cluster mean of everything else). Another way to look at it is by considering sets of articles. If we label the set of articles that generate  $c$  as the set  $C$ , and we label the set of all articles in the dataset as  $A$ , we can calculate  $e$  by generating the cluster mean of the articles in  $A - C$ .

We now have two vectors: one representing the cluster to be named, and another representing all the other articles in the dataset. If we compare components of the two vectors and calculate the difference of the two values we have a new vector  $d = e - c$ . This new vector  $d$  represents the difference of everything else and the particular cluster we want to name. By finding the component of  $d$  with the greatest value we will find the dimension of  $c$  that most differs from everything else in the dataset. This dimension corresponds to a unique word in our feature set that can be used to name the cluster of articles in set  $C$ .

### 5.2 Results and Performance

Since the vectors we used to represent the documents in our dataset were based on the titles of articles, we ended up with some really good summarizations of clusters. Our method chooses the most unique feature of a particular cluster, thereby eliminating the chance of naively naming a cluster based on the most frequent word in that cluster. By looking at the names generated for clusters, we noticed the absence of common words like "for", "the", and "a" which all are very frequent features over

our entire dataset. The reason why these aren't chosen as names of clusters is because they are uniformly distributed across all of the clusters. The only names applied to a particular cluster are chosen from words that are uniquely frequent to that cluster when compared to all the other clusters. By examining the figures presented in the previous section we see that this method works quite well for labeling clusters of articles.

## **6. Conclusions**

We were satisfied with the clusterer's performance in the newsgroup clustering tests, and the bootstrapped K-means approach gave us a relatively quick method to cluster large numbers of documents. The results of clustering articles proved more difficult to evaluate, but the visualization and summarization results are encouraging.

The scale of the intended task of clustering over 200,000 articles proved too great for the resources we had available. The memory and time required to cluster that many documents into a reasonable number of clusters was inhibitive. Further attempts to cluster on this scale need to be more efficient with space and/or algorithms to keep constant factors as small as possible. A better selection of features could reduce the iteration time significantly, although the identification of those features could take up more time than it saves. Despite all of these challenges with time and space, a system providing this functionality to the user would only have to cluster once, producing data files that could then be used to visualize and explore the results. New articles could be added to the appropriate cluster and the cluster mean recalculated. Thus, the up-front cost of generating an initial set of clusters and data files is great, but only a one-time penalty.

This type of clustering, visualization, and topic naming lays the foundation for some interesting applications. Future work could refine the clustering algorithm to include semantic or structural knowledge. Our goals include an algorithm that is domain-independent, so these additional heuristics should not depend on the content of the corpus. Once the clustering algorithm is finalized, the visualizations and topic headings could be used in a graphical browser of articles. Ideally, a user should be able to select a node from the cluster graph to expand and browse articles in that area. Finally, computing author and institution densities over the clusters could provide a simple and valuable extension.

## 7. References

Aas, Kjersti and Line Eikvil. “*Text Categorisation: a Survey*”, Norwegian Computing Center, 1999.

Joachims, Thorsten. “*A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization*”, Department of Computer Science, Carnegie Mellon University, 1996.

Lawrence, Steve, Kurt Bollacker, and C. Lee Giles. “*Indexing and Retrieval of Scientific Literature*”, Eighth International Conference on Information and Knowledge Management, CIKM 99, Kansas City, Missouri, November 2–6, pp. 139–146, 1999.

Manning, Christopher and Schütze, Hinrich. *Foundations of Statistical Natural Language Processing*, pp 495-528, 541-543. MIT Press, 1999.

Martin, Joel D. “*Clustering Full Text Documents*”, from the IJCAI-95 Workshop on Data Engineering for Inductive Learning. Institute for Information Technology, National Research Council Canada, 1995.

Nevill-Manning, Craig, Ian H. Whitten, and Gordon W. Paynter. “*Lexically-generated Subject Hierarchies for Browsing Large Collections*”, Stanford University and University of Waikato, 1999.

Noel, Steven, Raghavan, Vijay, and C.-H., Henry Chu. “*Document Clustering, Visualization, and Retrieval via Link Mining*”, Kluwer Academic Publisher, 2002.

Young, F.W. and Hamer, R.M. *Multidimensional Scaling: History, Theory and Applications*. Erlbaum, New York. 1987.