12-2006

TIC: Term Intersection Clustering of Text Documents

Casey R. Bartman
Grand Valley State University

TIC: TERM INTERSECTION CLUSTERING

OF

TEXT DOCUMENTS

Casey R. Bartman, B.A., M.D.

December 2006

This thesis is submitted in fulfillment of CS695, the Master's Thesis Research course in Computing & Information Systems.

Presented orally on December 14, 2006.

This thesis is dedicated to my wife,

Cindy, who encouraged my pursuit of this degree,

My sons, Casey Jr., Scott, Joey, Bradley, and Jeff

And

Stanley, Max, and Ozzie that kept me company

During my research.

ABSTRACT

Preliminary work performed by the author [3] investigated the clustering of text documents based upon the Boolean intersection of document term sets. In that algorithm, documents were associated with terms and the resulting sets of terms were intersected. If the intersection of the sets produced a set equal to or larger than a predefined minimum support level, that new set was considered a relevant cluster. The algorithm's first intersections were carried out at a three term level, allowing overlap of clusters at this level. Documents that were clustered were removed from further consideration and the process repeated at the two term level.

In this study the author's previously described algorithm was adapted to create a more robust and scalable implementation. The modified algorithm, Term Intersection Clustering or TIC, was evaluated and compared to the Bisecting K-Means algorithm. This comparison was performed employing the text of the bodies of articles that compose the Reuter's 21578 News Corpus [13].

While the cohesion, as defined and implemented by the author, was superior for the Bisecting K-Means algorithm, the actual value of the clusters, when physically reviewed was superior for the TIC algorithm. Run times were similar for the two algorithms. Furthermore, the data generated by the TIC algorithm was found to be superior for indexing and recall versus the output of the Bisecting K-Means algorithm.

INDEX

1.	Introduction	1
1.1	Programs Developed by the Author	3
2.	Preliminaries	4
2.1	Term Reduction	5
2.2	Vector Space Model	7
2.3	Cluster Similarity	8
2.4	Cluster Cohesion	11
2.5	Graphical Output	11
3.	Related Studies	12
3.1	K-Means Clustering	14
3.2	Bisecting K-Means Clustering	16
3.3	Hierarchical Agglomerative Clustering (HAC)	17
3.4	HAC Modifications	19
3.5	Frequent Term-Based Clustering	20
3.6	Mock's Algorithm	20
3.7	Bartman's Reverse Tree Clustering	21
4.	The Term Intersection Algorithm	23
5.	Project Method	24
5.1	System Specification	24
5.2	Data Set	25
5.3	Term Reduction—Selection	29
5.4	Clustering Methods	32
5.4.1	General Considerations	33
5.4.2	Term Intersection Clustering (TIC)	33
5.4.3	Bisecting K-Means Clustering	35
5.4.4	Cluster Cohesion	36
6	Results	37
6.1	Term Reduction--Selection	37
6.2	Term Intersecting Clustering	41
6.2.1	TIC GUI	44
6.3	Bisecting K-Means Clustering	49
6.3.1	Bisecting K-Means GUI	53
7	Discussion	54
7.1	Term Reduction—Selection	55
7.2	Term Intersection Clustering Versus Bisecting K-Means Clustering	56
7.2.1	Execution Time	56
7.2.2	Reproducibility	57
7.2.3	Cluster Quality	58
8.	Conclusions	62
9.	Future Study	64
	References	66

Equations

1. *tf-idf* 7
2. Minkowski Dissimilarity Metric 9
3. Sup Distance 9
4. Cosine similarity coefficient 11
5. Minkowski Dissimilarity Implementation 36
6. Cluster Cohesion 37
7. Cohesion Standard Deviation 37

Figures

1. Mock's GUI Output 12
2. Term Selection Versus Document Frequency 22
3. Document Data Object 27
4. Term Distribution 39
5. Documents Per Term Distribution 40
6. TIC Execution Time 41
7. TIC 3 Term Cluster Document Exclusion 42
8. TIC Clusters Formed Versus Support Level 43
9. Tree Structure of TIC GUI 45
10. Root screen image of TIC GUI 46
11. Example TIC GUI for Three Term Clusters 47
12. Example of TIC GUI Cluster Detail 48
13. Example of TIC GUI Document Detail 49
14. Clusters Formed As A Function Of Support For Bisecting K-Means 50
15 Cluster Cohesion Versus Support For Bisecting K-Means 51
16 Centroid Dimensions For Bisecting K-Means 52
17 Bisecting K-Means GUI 54
18 Gaussian versus Log-Normal Distributions 55

Tables

1. Term Distribution 39
2. TIC Results With Support Of 50 44
3. Centroids Versus Documents Non-Zero Dimensions 51
4. Bisecting K-Means Results 53

1. INTRODUCTION

The response to a query against the web or an enterprise's electronic data can overwhelm the user because it often includes thousands of documents such as articles, correspondences, and emails. For example, a recent search of Google Scholar for "*text clustering algorithms*" produced a result of 45,700 references. A corpus of this size does not allow practical extraction of relevant data by an end user without someway to organize the response. One commonly used approach to solve this problem is to automatically classify the returned documents into clusters such that similar documents are assigned to the same cluster, while dissimilar documents are assigned to separate clusters.

A major challenge in document clustering is how to define the notion of similarity and furthermore, how to apply it practically. A common approach is to identify a set of terms derived from the corpus of documents and then represent each document as a vector that reflects the presence and scale or the absence in the document of each one of the identified terms. Clusters are then created based upon similarity among documents as derived from the respective vectors of the different documents.

Several clustering algorithms have been proposed and evaluated in the literature. Two such algorithms are the *Bisecting K-Means* (a variation of K-Means) and the *Hierarchical Agglomerative Clustering* (HAC). The *Bisecting K-Means* algorithm is top-down, where by smaller clusters are created by repeatedly partitioning larger ones. In contrast, the *HAC* algorithm is bottom-up, where by large clusters are created by successively merging smaller ones. Empirical comparative studies reported in the literature show that while the

Bisecting K-Means algorithm is superior in terms of run time, the HAC algorithm has the advantage of producing better clusters.

The need to cluster textual data long proceeded the electronic age. In 1876, Melvil Dewey created the Dewey Decimal System to categorize or cluster text documents (books). The Dewey System [15] is a multilevel, hierarchical clustering system that divides documents into successively smaller clusters based on topics and subtopics. Similar clustering systems are utilized in Zoology to classify animals. Both examples represent the way in which humans tend to classify objects, a hierarchical manner that allows a user to focus on a small subset of the available data

Osinski [16] stated that "*popular search engines, as opposed to query-answering systems, return Web pages matching the user's question rather than the question's answer*". The consequence of this observation on search engines, and the large response generated to most queries, is that important answers to question may be concealed.

Another fundamental advantage of clustering is that it can reveal unexpected patterns in a corpus. If the user is unaware of the existence of these patterns, he would not initially pose a query to elicit their result. As a consequence, these patterns would remain undiscovered.

Work on language recognition has progressed slowly [22]. Because of this, current work on document clustering has primarily focused on treating documents as bags of words.

These algorithms have focused on clustering documents by the words or terms that the document contains without regard to the document subject matter.

The following thesis is organized as follows. Section 2 presents technical issues and background material related to document clustering. Section 3 reviews related studies in the field of text clustering, including the author's previous study. Section 4 presents the *TIC* algorithm. Section 5 examines the methods of implemented by the author for term selection, the *TIC* algorithm, the *Bisecting K-Means* algorithm, and cohesion calculation is presented. Section 6 presents the results of the implementations in Section 5. Section 7 discusses the implications of the results presented in Section 6. Section 8 contains the conclusions of this study with strategies for further study presented in Section 9.

1.1 PROGRAMS DEVELOPED BY THE AUTHOR

This study did not employ any stock programs. The author created programs that performed the following tasks in performing this study:

1. A program to parse the SGML files of the Reuters News Corpus.
2. A program to create the Java Document objects utilized.
3. A program to evaluate term distribution and frequency within the corpus.
4. A program to execute the *TIC* algorithm and GUI for the output.
5. A program to execute the *Bisecting K-Means* algorithm, a program to fix the data and prepare that data for the program to produce a GUI of the results of the algorithm.
6. Programs to calculate the cohesion of the results of the two algorithms and perform statistical evaluation on those results.

2. PRELIMINARIES

For even a small corpus of documents, the number of unique terms will be large. Methods have therefore been developed to reduce the number of terms used in clustering by removing irrelevant terms. A frequently applied method of reducing terms is by calculating the term's *tf-idf* value. In Section 2.1, the use of *tf-idf* values is examined as well as the role of stop words and term stemming.

Section 2.2 is a summery of the Vector Space Model of document representation employed by the *Bisecting K-Means* algorithm as well as by the *Hierarchical Agglomerative Clustering (HAC)* algorithm. The *TIC* algorithm does not make use of the vector space model.

The *cosine similarity* metric is summarized in Section 2.3. Both the *Bisecting K-Means* algorithm and the *HAC* algorithm frequently employ the *cosine similarity* metric for determining the similarity of documents. An alternative metric, the Minkowski Dissimilarity metric, is contrasted and compared to the *cosine similarity* metric in Section 2.3.

The method utilized in this study to determine the cohesion of the clusters formed by the clustering algorithms is outlined in Section 2.4. The test of the quality of the clusters formed by both the *Bisecting K-Means* algorithm and the *TIC* algorithm was the resulting cohesion of the clusters formed by the algorithms. As the implementation of the *Bisecting K-Means* algorithm, developed by the author, utilized the *cosine similarity*

metric, it was elected to employ the Minkowski dissimilarity metric in determining cohesion as presented in Section 2.4.

Section 2.5 contains a brief review of Graphical User Interfaces (GUI's) for representing data. While the purpose of this study is not to investigate GUI's for data representation, the review does provide insight into data representation. In particular, complicated GUI's such as Kohonen Maps [5] may not be the best way to represent large data sets. Images of the GUI developed to evaluate the clusters formed by the *TIC* algorithm and the *Bisecting K-Means* algorithm are reproduced in Figures 9 through 13 and Figure 17.

2.1 TERM REDUCTION

Even a moderately sized corpus of documents can contain a large number of distinct terms. The relatively small Reuter's corpus contains over 45,000 unique terms. Limited system resources makes utilization of this number of terms impractical. This constraint is secondary to the cost of storing and maintaining a complete term list. Furthermore, many common terms, also know as stop words, have little value in the clustering of documents. Therefore, a metric to remove terms of little perceived value in the clustering process is required. The common designation for this class of metrics is Term Reduction.
Salton [18] [19] proposed a five step process to affect term reduction.

1. Identify terms within a corpus.
2. Remove stop words.
3. Perform a stemming type operation.
4. Calculate term weights for the remaining terms.
5. Represent each document in the corpus as a weighted term vector.

The first step is self explanatory and will not be discussed further. The final step addresses weighted term vectors. As the *TIC* algorithm is Boolean, this step also is not considered further. Steps 2-4 deal with decreasing the number of terms identified in step one, to a workable number.

In step two, each term is compared to a dictionary of common terms, frequently referred to as stop words. Terms that are contained in the dictionary are removed from further consideration. Several lists of stop words are available for use [8]. Potential limitations on the use of stop words have been outlined by Baeza-Yates [2]. Furthermore, common terms will be identified by calculating term weight (see below).

In step three, suffixes and possibly prefixes are removed from terms to create a root term. Salton [18] [19] noted the limitations of the use of stemming algorithms, specifically, the relevance of terms to certain data sets. He further noted that "*term reduction to four or five charaters provided almost as good discrimination between relevant and nonrelevant documents*".

In step 4 of Salton's metric the weights of terms are calculated. A frequently applied metric to perform this calculation is the *tf-idf* metric [19]. The rationale for this metric is two fold. First, for a term to be relevant to clustering, it should only occur in a selected number of documents. Terms that occur in a large percentage of the documents in a corpus are of little value in the clustering process. Second, a term that occurs frequently

in a given document should be significant in identifying and distinguishing that document for other documents contained in the corpus.

Equation 1 denotes the formula for the *tf-idf metric.* For a term to satisfy condition one above ,the *–idf* should also be maximized (i.e. few documents should contain term *i*). To satisfy the second condition, the value for tf_{ij} should be large (i.e., there should exist a high frequency of term *i*, in document *j*).

$$w_{ij} = tf_{ij}\left(-\log\left(\frac{n}{N}\right)\right)$$

Where w_{ij} is the weight of term i in document j, tf_{ij} is the frequency of term i in document j, n is the number of documents within the corpus that contain term i, and N is the total number of documents in the corpus.

Equation 1. *tf-idf* term weight.

2.2 VECTOR SPACE MODEL

Both the *Bisecting K-Means* algorithm and the *HAC* algorithm represent documents, as a vector in *n* dimensional space, where *n* is the number of terms that are used in the clustering process. The set of terms **Γ,** a set of *n* terms to utilize for clustering, is selected by a method as described in Section 2.1.

Each document **D** is abstracted to a set of terms ***D*** were $D \subseteq \Gamma$. Furthermore, as a consequence and desire of the selection of **Γ**, for any given document, the number of terms in ***D*** that are elements of **Γ**, will be significantly less than the total number of terms in **Γ**. A document may then be represented by a vector $V = [w_1, w_2, \ldots, w_n]$, were w_1, w_2, $\ldots, w_n$ are the weighted values for the terms $t_1, t_2, \ldots, t_n$ that are defined as

$\{t_1, t_2, ..., t_n\} \subseteq \Gamma$, where the weighted value are calculated utilizing the *tf-idf* metric described in Section 2.1. To simplify calculations elaborated on in Section 2.3, the length of V is usually normalized such that $|V|=1$.

Terms employed in the clustering process will only occur in a limited number of documents. The value for terms that do not occur in a document, but are elements of **Γ**, will be zero. Therefore, the vector that represents a given document will consist of attributes primarily equal to zero and a limited number of attributes that have a value $0 < a \leq 1$ (where $|v| = 1$).

A limitation of the vector space model is the number of attributes that have a value of zero. While the size in memory of a term vector may be reduced by utilizing a sparse representation of that vector, the weighted values of each term in the vector must be considered in calculating similarity. Methods of determining similarity or dissimilarly are evaluated in Section 2.3.

2.3 CLUSTER SIMILARITY

Cluster pairs are compared for similarity based on one of several methods. The primary method utilized with the vector space model to compute the similarity of two vectors is the *cosine similarity* metric [9].

An alternative to the *cosine similarity* metric is the Minkowski [9] dissimilarity metric. The Minkowski metric (Equation 2) calculates the difference between vector models. As

the value is determined by subtracting the magnitude of the given dimensions of the vector model, every dimension is significant in the determination of the dissimilarity value. This determination is similar to a logical *or* statement, if either vector has a magnitude for a given dimension, then that dimension is significant. The greater the value calculated by the Minkowski metric, the more dissimilar the vectors.

$$d(i,k)=\left(\sum_{j=1}^{n}\left|x_{ij}-x_{kj}\right|^{r}\right)^{1/r}$$

Where d is the dissimilarity between two vectors, n is the number of dimensions in the vectors, x_{ij} is the magnitude of dimension j in vector i, x_{kj} is the magnitude of dimension j in vector k, and r *is* an integer >0, selected by the user.

Equation 2. Minkowski dissimilarity metric.

In the Minkowski metric as the value of r increases the weight of the most dissimilar attributes increase. A special case called the "sup" distance, occurs as $r \to \infty$, where only the greatest difference in attribute values is considered (Equation 3).

$$d(i,k)=\max_{1\le j\le n}\left|x_{ij}-x_{kj}\right|$$

Equation 3. The "sup" distance.

Two other special cases are when $r = 1$ and $r = 2$. The prior case has been referred to as the Manhattan distance [9]. In this case the sum of the difference in attribute values is considered. The name Manhattan distance has been given to this case as it simulates the distance between points when walking the blocks of a city (i.e. the non straight line distance). The later case, $r = 2$, produces the Euclidian distance difference, or straight line distance.

The *cosine similarity* metric is a commonly applied method of determining the similarity between two term space vectors. The denominator of equation 4 normalizes the length of the term vectors under consideration to one.

The numerator of Equation 4 differs from the Minkowski metric in that the products of the magnitudes of the two vectors' dimensions are summed instead of the differences. For a vector dimension to be significant, both vectors must have a non zero value for that dimension. Because of this, a value of zero is given a Boolean false quality, and that attribute is not considered in the similarity of the two vectors. This determination is similar to a logical *and* statement, if both vectors have a magnitude, then that dimension is significant.

The limitation of this can be visualized by evaluating three vectors in two-dimensional Space:

$$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \text{and} \quad z = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

The cosine similarity of x, y, and z is zero. The Minkowski dissimilarity for x and y, where r=2 in equation 2, is $\sqrt{2}$, and the Minkowski dissimilarity for x and y with respect to z is one. Intuitively, there is a difference in the dissimilarity between vector x and y with respect to each other and their dissimilarity to the null vector z. (The null vector z is assumed to have a length of 1 for purposes of Equation 4).

$$\cos ine = \frac{d_x \bullet d_y}{|d_x||d_y|} = \frac{\sum_{t=1}^{t=n} d_{xt} * d_{yt}}{\sqrt{\left(\sum_{t=1}^{t=n} (d_{xt})^2\right)} * \sqrt{\left(\sum_{t=1}^{t=n} (d_{yt})^2\right)}}$$

Where dx and dy are document term vectors and t is the weighted values of the terms utilized in clustering.

Equation 4. Cosine similarity coefficient.

The run time to calculate the cosine similarity of two document vectors will be O(n), were n equals the number of terms utilized in creating the document vectors.

2.4 CLUSTER COHESION

The cohesion of a set of document vectors is a metric to determine how similar the elements of a cluster are to the cluster centroid. Cohesion has frequently been employed to judge the quality of the clusters formed by various algorithms. The metric advanced and implemented by the author is discussed in Section 5.4.4. As noted in Section 5.4.4, cohesion was determined by use of the Minkowski dissimilarity metric. Therefore, in this study, the smaller the calculated value for cohesion, the more similar the document vectors are to the cluster centroid.

2.5 GRAPHICAL OUTPUT

The concept of representing the clustering results graphically is appealing. Chen [5] found that while a Kohonen Map's graphical interface was appealing to users, it was most beneficial for broad categories of data were the user skipped around between categories. The Kohonen Map did not work well for users who wanted to do directed research. Furthermore, when evaluated on test subjects, some desired the more traditional hierarchical graphic output.

An example of the more traditional hierarchical graphic output was presented by Mock [14] in his study as seen in Figure 1. As the GUI's in this study were used to evaluate the output of the clustering algorithms, they were of the traditional hierarchical appearance.

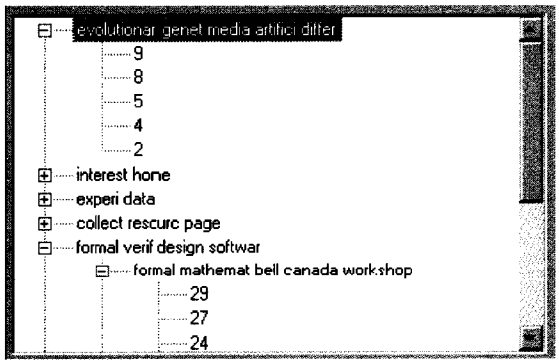


Figure 1--Mock's GUI Output

3. RELATED STUDIES

Two major solutions to clustering have been evaluated in the literature [7]. The first is partitioning clustering, where the corpus is repeatedly divided into smaller clusters. Studies by Steinbach [20] have demonstrated that a variation of this method, the *Bisecting K-Means* algorithm, produced clustering results as good as the second solution, *Hierarchical Agglomerative Clustering* algorithm (HAC). As the *Bisection K-Means* algorithm is defined as a linear clustering algorithm, with respect to the number of documents to cluster, it has been determined to be superior to *HAC* algorithms which run in $O(n^2)$ to $O(n^3)$ time, where n equals the number of documents to cluster. Steinbach's conclusions have made the *Bisecting K-Means* algorithm the *Gold Standard* by which clustering algorithms are judged.

The general *K-means* algorithm and the *Bisecting K-Means* algorithm are evaluated in Sections 3.1 and Section 3.2 respectively. Limitations on *K-Means* algorithms are evaluated, including the apriori need to know the number of cluster to form. This limitation can limit the ability for a clustering algorithm to discover unknown relations within a corpus.

In Section 3.3, the other method of document clustering that has received significant study is reviewed, the *Hierarchical Agglomerative Clustering* algorithm (HAC). In the *HAC* algorithm, each document in the corpus is considered as a singleton cluster. The two most similar clusters are merged into one cluster and that cluster is added back into the cluster pool for the next iteration of the clustering process.

The algorithm will continue until the entire corpus is clustered into one large cluster, the original corpus. Termination conditions must therefore be set to halt the algorithm prior to reaching the above end point.

Another limitation of the HAC Algorithm is the run time. Depending on the linking of the clusters, this can be of $O(n^2)$ or $O(n^3)$.

Section 3.5 evaluates clustering based on itemsets. In the algorithms evaluated in this section, clustering is performed by generation of an itemset or termset lattice. The results of the study revealed clustering superior to a 9-secting k-means algorithm, but inferior results to a *Bisecting K-Means* algorithm. Limitations of the study by Beil [4] were

incomplete information on the term set chosen and the 9-secting k-means algorithm used in the comparison.

In Section 3.6 the Intel Technical Report by Mock [14] is evaluated. This algorithm utilized a variation of the itemset lattice algorithm where clustering at a three term level was performed initially. Reasonable results were obtained with this algorithm. Furthermore, a variation was presented that allowed overlapping clusters, a significant factor in this author's opinion. Limitations were the use of a small term set and limiting clustering to a three term itemset lattice. Furthermore, Mock utilized a small, non-standardized term set in his evaluation. Mock's report was the initial impetus for the study by Bartman [3] and this further evaluation.

Section 3.7 reproduces the authors work on the Reverse Tree Clustering Algorithm presented at DMIN06. In this study the author developed and refined a preliminary implementation of the algorithm proposed by Mock. This algorithm demonstrated reasonable quality of clustering and reasonable run times, when performed on the titles of the Reuter's 21578 News Corpus articles.

3.1. K-MEANS CLUSTERING

The *K-Means* algorithm is a frequently employed for clustering. Briefly, the algorithm proceeds as follows. As a preliminary step the algorithm requires the user to supply the number of desired clusters. The user must therefore have apriori knowledge of the number of clusters that exist in a corpus. Next, a set of centroids (vectors) are selected, to represent the central value of each cluster. The number of centroids selected is equal to

the number of clusters to form. The initially selected centroids frequently are randomly selected documents form the given corpus. The algorithm then proceeds as follows [20].

1. Documents are assigned to the cluster with centroid most similar to a given document.
2. The values for the centroids are re-calculated.
3. Steps 1 and 2 are re-iterated until there is no change in the cluster composition, or a predetermined number of iterations occur.

The concept that *K-Means* algorithms are linear is a misnomer. As Zamir [25] demonstrated, the *K-Means* algorithm runs in O(*nk*T) time were *n* is the number of documents to cluster, *k* is the number of clusters to form, and T is the number of iterations to perform.

While the *K-Means* algorithm has been described as a linear algorithm with respect to the number of documents in a corpus, there are other significant variables in the determination of the algorithm's run time. Zamir [25] demonstrated that the run time of the *K-Means* algorithms is a linear function of the product of the number of documents within the corpus, the number of clusters to form, and the number of iterations used to refine the cluster composition. Bartman's [3] study on the Reuter's News Corpus demonstrated the formation of 793 overlapping clusters utilizing clusters with a minimum support of 20 documents. This revealed that the number of clusters to form is a significant number compared to the total number of documents in the corpus.

This author contends that the run time for the *K-Means* algorithm is actually O(*nkt*T). The values for *nk*T are as noted by Zamir, the value T is equal to the number of terms contained in the document vectors. The inclusion of T is warranted since, during each iteration, each document vector must be compared for similarity to the centroid of each cluster. As noted in Section 2.3, the run time for the cosine similarity metric is O(n).

Bartman's evaluation of the Reuter's News corpus utilizing overlapping clusters revealed a cluster count of the magnitude of 10^2. The number of terms used in the clustering algorithm was also of a magnitude of 10^2. If the number of iterations to perform a *K-Means* clustering is of a magnitude of 10^1, then the magnitude of the value *kt*T above will be 10^5. As the magnitude of the number of articles in the Reuter's corpus is 10^4, this author believes that to call *K-Means* clustering a linear run time is a misnomer.

Another difficulty of the basic *K-Means* algorithm is that there is no overlapping of the clusters. For text articles such the Reuter's News corpus where many of the articles deal with financial matters, this represents a serious limitation. The author also believes that this is a serious limitation for the clustering of results from a web search engine query. Variations of the *K-Means* algorithm, *Fuzzy K-Means* Algorithms [10], do address this issue.

3.2. BISECTING K-MEANS CLUSTERING

The *Bisecting K-Means* algorithm is a variation of the *K-Means* algorithm, in which the corpus is divided into smaller clusters until the given number of desired clusters is

derived. In the *Bisecting K-Means* algorithm a cluster (the whole corpus in the first iteration) is chosen to split into two clusters. Two centroids are chosen, and two new clusters are created from the initial cluster, in the manner of the above *K-Means* method. The *Bisecting K-Means* algorithm has been summarized by Steinbach [20] as follows.

1. Pick a cluster to divide.
2. Find two sub-clusters utilizing the basic K-Means algorithm.
3. Iterate step 2 for maximum similarity.
4. Repeat steps 1 through 3 until the predetermined number of clusters is derived.

Steinbach [20] states that the *Bisecting K-Means* algorithm is linear in run time with respect to the number of documents. While this is true for small corpuses, this author does believe that the number of terms, the number of iterations to perform, and the number of clusters to form are significant and distort the run time significantly. Steinbach [20] further demonstrated results similar to *HAC* algorithms utilizing a *Bisecting K-Means* algorithm. He summarized that the *K-Means* algorithms were superior as they have been called linear time clustering algorithms.

3.3 HIERARCHICAL AGGLOMERATIVE CLUSTERING (HAC)

Voorhees [21] and Willett [23] proposed a hierarchical agglomerative clustering algorithm for document retrieval. In the *HAC* algorithm each document in the corpus is initially treated as a singleton cluster. The clusters are compared, and the two most

similar are merged to form a new clusters. The process is then repeated until a termination condition is reached or only one cluster remains.

The limitations of the HAC algorithm have been well documented [20]. These limitations include the lack of overlapping clusters, the formation of clusters with unrelated documents, and slow run times, $O(n^2)$ or $O(n^3)$, versus $O(n)$ or linear run times for the *Bisecting K-Means* algorithm.

The lack of overlapping clusters is a function of the algorithm. Unfortunately, a document may well belong to two or more distinct clusters.

The formation of clusters of unrelated documents may be the byproduct of the underlying algorithm, or the structure of the data. Without a termination condition, the *HAC* algorithm will eventually create one large cluster of the entire corpus. Two solutions exist that address this problem. The first is to define a minimum number of clusters to be form. The second is to stop the algorithm when the dissimilarity within a cluster exceeds a predefined limit.

Unrelated documents may also be included in the same cluster by the data patterns. As singleton clusters are merged, further comparison of cluster similarity is based on some function of the singletons that make up the aggregated cluster. This method is usually to calculate a centroid for the cluster. The possible errors and limitations were well documented by Karypis [11].

If a user has apriori knowledge of the corpus, then the number of clusters to form may be known. If such knowledge is not available, a choice of the number of clusters to form is at best an educated guess.

The run time of the HAC algorithm is $O(n^2)$ for single-link clustering and $O(n^3)$ for complete-link clustering, where n is the number of documents to cluster. This has been found to be unsatisfactory upon review of clustering of relatively small corpus [20]. Jain [9] provides a detailed definition of single-linked and complete linked clustering. Single-linked clustering is a metric where elements are merged into clusters by comparing the similarity of the elements. Clustering continues until each element is included into a cluster of the whole. Complete-link clustering compares the similarity of every element, even after it has been included into a given cluster. For a corpus of five elements, single-link clustering will require four iterations $(n-1)$, while complete-linked clustering will require ten iterations $\left(\sum_{i=1}^{i=n-1} i\right)$ [9].

3.4 HAC MODIFICATIONS

Zamir [24], et. al, reviewed a modification of the traditional HAC algorithm called the Word-Intersection Clustering algorithm (Word-IC). While the Word-IC algorithm produced more cohesive clusters, than single-linked HAC algorithms, it again ran in $O(n^2)$ time, where n represents the number of documents to cluster.

Zamir then proposed a further modification, the Phrase-IC algorithm. This algorithm produced clusters of tight cohesion like the Word-IC algorithm, but only ran in O(n log n) time, where n is the number of documents to cluster.

The Phrase-IC algorithm utilizes a sufflx array to perform clustering [2]. While the Phrase-IC algorithm ran in O(n log n) time, this does not take into consideration the high construction cost of creating the suffix array.

3.5 FREQUENT TERM-BASED CLUSTERING

Beil [4] identified the limitations of current clustering algorithms. He proposed two clustering algorithms based on frequent term set to evaluate the similarity among documents. The first algorithm, *Frequent Term-Based Clustering*, creates a flat clustering of the entire corpus. The second algorithm, *Hierarchical Frequent Term-based Clustering*, creates hierarchical clusters. A method to reduce overlapping of the clusters was utilized. His results, utilizing a termset lattice, demonstrated similar quality of cluster formation versus a 9-secting k-means algorithm, but inferior results to a *Bisecting K-Means* algorithm.

Beil does not elaborate on the size of his term set, the implementation of the 9-secting k-means algorithm, or on his methods of term selection.

3.6 MOCK'S ALGORITHM

Mock [14] proposed an algorithm based on Word Intersectional Hierarchical Agglomerative clustering. In this algorithm, clustering was initiated at the specific level and progressed to a more general level. The algorithm used a term list of 100 terms. Candidate terms for each document were generated utilizing *tf-idf* values. To be a successful candidate, a term's *tf-ifd* value had to satisfy $.05 \leq tf\text{-}idf \leq 1$. The generated

pool of candidate terms was then ordered by the term's document frequency (df). Terms with a document frequency > 25% or with an absolute occurrence of less than three documents were dropped. As the corpus employed by Mock consisted of 50 documents, the range of df's of the candidate set of terms was 4 %< df <25 %. A final list of 100 terms was generated.

Mock also proposed a non-overlapping algorithm and an overlapping algorithm. The non-overlapping algorithm was not further evaluated. In the overlapping algorithm, clustering was initially carried out by creating clusters of documents that matched rule sets generated utilizing three terms. The number of rule sets to be evaluated was expressed by $\binom{n}{d}$ where n=100 and d=3. For a list of 100 terms this required the evaluation of 161,700 rule sets. Each cluster was then evaluated for additional terms that existed in each document. A new set of clusters was then generated combining the results of the above clustering as clusters along with singleton clusters of documents that were not clustered above. The above process was then repeated utilizing rule sets derived from two terms. A final iteration was then performed utilizing one term rule sets.

Limitations of this algorithm were the construction of three term clusters and the large number of rule sets that had to be evaluated even for a modest set of 100 terms.

3.7 BARTMAN'S REVERSE TREE CLUSTERING

An initial study by Bartman [3] on clustering of the Reuter's corpus titles demonstrated that the number of clusters formed was a function of the number of terms selected for the clustering process. Furthermore, the degree of overlap or fuzziness of the clusters was a

function of the minimum cluster size. The number of unique terms considered in the clustering of the titles of the corpus was 14,228. The total number of terms was 137,424. Run time for the selection of terms was approximately 60 seconds.

Term selection was performed by evaluating the document frequency of terms in a graphic representation as represented in Figure 2. A maximum *idf* of 2.56 was selected for a term to be considered a critical term. An *idf* of 2.56 in Figure 2 for the terms in the titles of documents in the Reuter's Corpus represents the occurrence of a given term in at least 60 documents. The relationship of the number of terms selected and the word threshold tended to be linear for word thresholds of 60 or greater as demonstrated in Figure 2. For word thresholds of less than 60, the relationship of the number of terms selected, and the word threshold tended to have an inverse exponential relationship.

The full text of Bartman's study is reproduced in Appendix C.

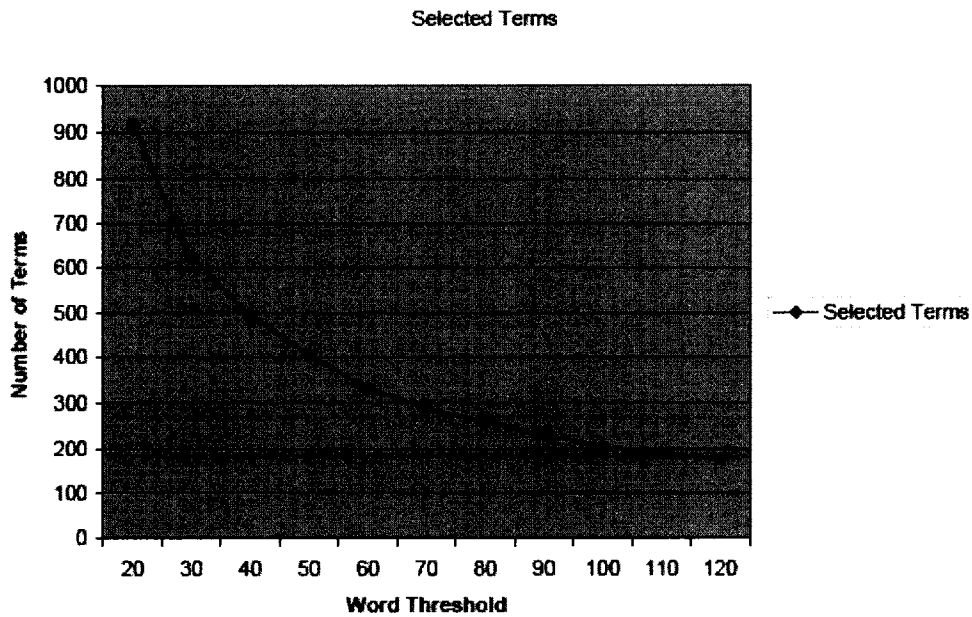


Figure 2—Term selection versus document frequency.

4. THE TERM INTERSECTION CLUSTERING ALGORITHM

The focus of this thesis is the *TIC* algorithm. The *TIC* algorithm is an evolution of the Reverse Tree Algorithm, discussed in Section 3.7 and Appendix C. The *TIC* algorithm consists of the following nine steps.

1. Select the critical terms in the corpus to be cluster.
2. Perform a weight metric on the terms. Remove terms below a set minimum weight and above a set maximum weight.
3. Represent individual documents as a set of critical terms contained within the given document such that $D = \{t_i, t_j, ..., t_n\}$.
4. Create clusters of documents such that all documents in a cluster have three terms in common, by means of a frequent itemset metric, such that $C = \{D_i, D_j, ..., D_n\}$ where:

   $t_i \in D_i,\ t_i \in D_j,\ t_i \in D_n,\ ...,\ t_n \in D_i,\ t_n \in D_j,\ t_n \in D_n$

   Iterate through all possible three term combinations.
5. Remove documents from further consideration that cluster at the three term level.
6. Repeat step 4 for documents with two matching terms.
7. Remove all clustered documents from step 6.
8. Form a cluster of documents with only one term.
9. Form a cluster of documents with no matching terms.

5. PROJECT METHOD

Evaluation of the *TIC* algorithm and the *Bisecting K-Means* algorithm required the development and implementation of multiple Java programs by the author. The SGML files that composed the Reuter's News Corpus had to be parsed into workable objects for the clustering process. The identification of terms within a document was included with the parsing and extraction of the documents from their original SGML file.

Next, Java programs had to be developed to implement both the *TIC* and the *Bisecting K-Means* algorithms. Timers had to be included within these programs to evaluate run times. Further, a custom implementation of the *Bisecting K-Means* algorithm had to be developed to allow subsequent evaluation and comparison to the *TIC* algorithm.

Initial evaluation of the results of the execution of the two algorithms was accomplished by creating Java programs that would calculate the cohesion of the clusters formed as well as the range and standard deviation of those clusters. These were separate programs from the implementation of the algorithms, so as not to affect the execution time of the algorithms.

Finally, a GUI was created by the author to allow manual inspection of the clusters formed. This required pre-processing programs of the results of the algorithms and the creation of an actual GUI programs.

5.1 SYSTEM SPECIFICATION

All results recorded in this study were performed on a single channel AMD 200MHz Sempron system with 1028MB of DDR ram running Red Hat Fedora Core 4.

All java code was written and then compiled using Sun Java 1.4.2. During this study Java 5.0 was introduced by Sun. As this version of Java introduced the Element attributes to the Set interface, this version of Java was not adopted.

5.2 DATA SET

This study evaluated the implementation of an Apriori clustering algorithm utilizing frequent term itemsets. The Reuter's 21578 News Corpus was used as the data set for the evaluations.

The Reuter's corpus consists of news articles reported by the Reuter's News Service in 1988. The corpus is not a complete compilation of articles, but a selection of articles from the Spring, Summer, and Fall of 1988. This collection of articles was originally compiled, indexed, and categorized by the staff of Reuters [13]. Based on recommendations made at SIGIR 96, Finch and Lewis [13] removed duplicate documents and formatted the collection with SGML tags.

The first tag for each document was <REUTERS>. This tag has five attributes, two of which were significant for this study. The first attribute of note was that of TOPIC. A YES value for the TOPIC attribute indicated that in the original collection of Reuter's documents at least one entry was made under topics. Only articles that had a TOPIC attribute of YES were considered in this study. The second attribute of consequence was the NEWID attribute. This attribute was a unique integer value that was assigned to each

document. This value was used as a reference to each document during performance of the algorithms.

The next two tags of significance for this study were the <TOPICS> tag and the <PLACES> tag. These tags enclosed a list of topics and geographical locations respectively that the individual article had been assigned previously. Only articles that had values recorded under topics and or places were considered for clustering.

The next tag utilized was the <TEXT> tag. The <TEXT> tag had the attribute TYPE. The final two tags were contained within the <TEXT> tag, they were <TITLE> and <BODY>. The <TITLE> tag enclosed the title of the article and the <BODY> tag enclosed the text of the article. The text enclosed within the <BODY> tag was used to perform the clustering algorithms.

As the intention of this study was to compare and contrast the *TIC* algorithm and the *Bisecting K-Means* algorithm, it was elected to extract each document from the document's original SGML file and create a Java object of that document. As noted, the author developed Java programs to extract the documents and create the document objects. Each document object (Figure 3) was saved for use in the *TIC* algorithm and the *Bisecting K-Means* comparison. The time to create and store the document objects was not included in the run time of either algorithm.

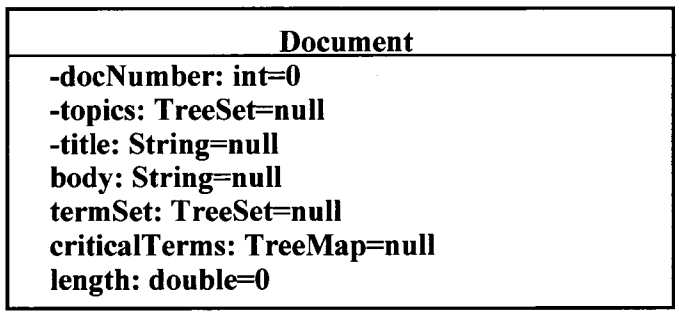


Figure 3. Java object structure of a document.

The structure of the document objects was a set of attributes only. A simple constructor method assigned an identifier value for the given document. Besides an integer identifier, the other attributes recorded for each document were the following.

Two String attributes were contained within the document objects. The first string contained the title of the document as assigned by Reuters. The second string contained the text body of the document. Neither of these attributes were directly employed in the clustering process. Both of these attributes were utilized in the GUI developed for the *TIC* algorithm and for manual evaluation of the quality of the clusters created by the *TIC* algorithm and the *Bisecting K-Means* algorithm.

A TreeSet attribute containing the contents of the Topics tag and the Places tag, as extracted from the original SGML file, for the given document was created. The intended use for this attribute was the calculation of F values [4] of the clusters formed by the *TIC* algorithm and the *Bisecting K-Means* algorithm. The values that were recorded in the original SGML files, on manual inspection, had little intuitive value in evaluating a document's correct fit within a given cluster. Because of the limited value of the Topics and Place tags, the F-value were not considered in the evaluation of the clusters formed.

A TreeSet attribute containing a list of all terms within a document after stop words have been removed and the terms have been stemmed was developed. This TreeSet was not utilized in either the *TIC* algorithm or the *Bisecting K-Means* algorithm. It was employed in determining the results of the *idf* metric and critical term selection.

A TreeMap containing terms selected for the clustering algorithms, hence forth the critical terms was produced. The keys for the TreeMap were those critical terms that occur in the document, and the values were the number of occurrences of that key in the document. The Bisecting K-Means algorithm primarily employed this TreeMap. The *TIC* algorithm primarily made use of the set of keys.

The final attribute of the document object was a double number containing the Euclidean length of a term vector generated for the document. This was utilized by the *Bisecting K-Means* algorithm to normalize the attribute values in the preceding TreeMap to a document length of one. Furthermore, cohesion testing of the clusters formed by the *Bisecting K-Means* algorithm and the *TIC* algorithm employed this value. The *TIC* algorithm did make use of the Euclidean length value and critical term occurrence value for those critical terms that had an occurrence of less than two for documents with an Euclidean length of more than six (i.e. those terms were dropped from the document set of critical terms).

A final set of 19,042 documents were selected for the study. All documents that contained attributes for the topics and or location SGML tags were included in the set.

5.3 TERM REDUCTION & SELECTION

Term reduction in this study was separated from algorithm implementation. This removed any bias for either algorithm in the given algorithm's implementation.

All of the evaluated algorithms in Section 3 required a method of term selection and reduction. A frequently employed method was the calculating of the product of the document term frequency (*tf*, the number of documents that contain a term) times the negative log of the term document frequency (*-idf*) in the corpus.

The *TIC* and *Bisecting K-Means* algorithm, as well as those algorithms discussed in Section 3, all treat text documents as bags of words. The Reuter's 21578 News Corpus consisted of a large number of terms, and the final set of terms incorperated in clustering was derived as outlined below.

Initial evaluation of 19,042 documents in the Reuter's 21578 News Corpus, which had been classified by topic or location, revealed 45,691 unique terms prior to the consideration of any stop words or stemming. A method of term reduction was therefore mandated for both the *TIC* algorithm and for the *Bisecting K-Means* algorithm.

In this study, stemming was performed after identifying a term list. An evaluation of the algorithm developed by M.F. Porter [17] resulted in an unsatisfactory set with prolonged run times. Salton's [18] [19] metric of stemming terms to the first five characters was

then evaluated. This metric minimally increased the time to select the terms from the corpus, and resulted in the development of a satisfactory term set.

Having developed a candidate term set, stop words were removed. This was accomplished by a manual review of the candidate set. While not adopted, a GUI interface could be developed that would expedite this process for an end user. The manual review of the candidate set was accomplished by the author developing a program that produced an output file with the individual terms and those terms' document frequency. Section 6.1 elaborates on the unexpended distribution within the Reuter's Corpus.

A review of the methods of term selection in the algorithms reviewed in Section 3 revealed limited descriptions of the operation of the *tf-idf* metric in term selection. From review, it appeared that most authors calculated the document frequency of a term first. This value was represented by the number of documents that a term occurred in divided by the total number of documents. By determine the –log of the preceding value, one was left with a value, *–idf,* the range of $0 \leq$-idf$\leq \log$ (number of documents).

A value of 0 represents a term that occurred in every document and a value of log (number of documents) represented a term that only occurred in one document in the corpus. Mock in his algorithm narrowed the range to $-\log\frac{1}{4} <$-idf$< -\log\frac{1}{25}$.

Unfortunately, there was no recommended range of *–idf* values, and as in Mock's case, the values were determined to create a final term set of appropriate size.

The second portion of the *tf-idf* metric is the frequency of a term in a given document. While various authors state that *tf*idf* values were incorporated to select a set of terms for clustering, exactly how the *tf*idf* values were utilized were not elaborated. The presumed hypothesis is that a relevant term will occur frequently in a subset of the corpus and will result in a term frequency of zero for most documents (i.e. a relevant term will only occur in a few documents in the corpus) for a given term. Unfortunately, in the studies reviewed, the method by which the *tf*idf* metric was utilized was poorly defined.

Salton's [19] original metric had a term set developed for each document based on a non-defined minimum *tf* value within each document. The document frequencies and subsequent *–idf* values for terms were based on the preceding set of terms. Salton did not provide insight into the values to utilize for ranges for the *tf* value nor the *–idf* value.

The impression from review of methods is that the *-idf* values for a given term were employed to select the final set of terms to use in clustering, and that the *tf*idf* was utilized to weight the terms in calculating the document vectors in step five above.

The use of term *-idf* does make the use of stop words redundant, for the removal of common terms. If a stop word was defined as a term that occurs frequently and is considered of little value in the clustering process then that term should have a low *-idf* value. Any term that occurs frequently, will have a low *-idf* and therefore will be identified when the *-idf's* of terms are calculated. Furthermore, a stop word from a pre-complied dictionary may be relevant to a given corpus and will be identified by

calculating the *-idf's* of the entire term set. Considering the above and also the noted limitations of stop words, their use is called into question.

Therefore, the final term set was selected by consideration of all terms that occurred in more than 5% of the documents and in less than 20 % of the documents. These terms were stemmed as noted above, and a list of stop words developed. This was accomplished by a program implemented by the author. This list consisted of 128 stop word terms, of which, 90 stopped terms occurred in less than ten per cent of the documents. In addition, eleven terms had custom stems developed (Appendix A).

The term set developed consisted of 182 terms (Appendix B). Furthermore, only 21 documents out of 19,042 documents contained none of the terms. The maximum *-idf* value was 1.29 for the stemmed term *agric* and the minimum *-idf* value was 0.71 for the stemmed term *state*. Except for the five non-stopped terms, the minimum *-idf* value was greater than 0.60 for all non-stopped terms. Section 6.1 evaluated this topic further..

To calculate the list of terms and their associated *tf-idf's* two passes over the corpus were required as noted by Larsen [12].

5.4 CLUSTERING METHODS

As the goal of this thesis was to present the *TIC* algorithm and compare it to the well-established *Bisecting K-Means* algorithm, the author, to execute both algorithms, developed Java programs. Furthermore, a set of programs were developed to evaluate the results of the execution of the algorithm programs.

5.4.1 GENERAL CONSIDERATIONS

During the evaluations of the *TIC* algorithm and the implementation of the *Bisecting K-Means* algorithm, a support level of 50 was established for valid clusters. Support was defined as the minimum number of documents that had to exist in a primary cluster level for that cluster to be valid. If a cluster did not contain a number of documents equal or greater than the level of support, the cluster was not formed, and the documents continued to be processed as if they had not been clustered. The support level of 50 was utilized both in the execution of the *TIC* at the three term level and the *Bisecting K-Means* algorithms. Both algorithms could function with lower support levels. Unfortunately, at lower support levels on the described system, memory was insufficient to create GUI's for the results.

5.4.2 TERM INTERSECTION CLUSTERING (TIC)

The algorithm presented in this study reduced each document to a set of critical terms from the corpus that may be defined as:

$$d_i = \{t_a, t_b, ..., t_x\}$$

Were d_i is document i, and t_a, t_b, t_x are terms in the set of critical terms.

For a term to be included in a document's definition, Section 4, step 3, the given term had to be present in the TreeMap, of critical terms, for the given document. Terms were weighted, Section 4, step 2, in a Boolean manner. If the length of the critical term set was greater than 6, the given term had to have more than one occurrence in the document.

Clusters were then formed by creating sets of documents that had, three, two, one, or no. terms in common.

Documents were allowed to exist in more than one cluster at a given level. This introduced Fuzziness into the algorithm. This was believed to be critical to text clustering as a review of a sample of documents supported this assertion.

The primary level of clustering was three terms (i.e. all documents in a cluster set had to have three critical terms in common). If a document clustered at the three term level, it was eliminated from consideration in cluster formation at the two term level. Similarly, documents that cluster at the two term level are not included at the one term level.

Cluster formation was evaluated at the four term level. With a support level of 20 documents for successful clustering, 12,634 documents out of 19,042 or 66 per cent did not cluster at the four term level. Therefore, clustering beyond the three term level was not further considered.

The clusters formed by the algorithm were indexed by the terms that defined the given cluster. This index was incorporated in the creation of a GUI for manual evaluation of cluster quality. As noted in Section 5.4.1, support for a cluster to be valid was defined to be 50 or approximately .25% of the total number of documents in the corpus.

5.4.3 BISECTING K-MEANS CLUSTERING

An implementation of the *Bisecting K-Means* algorithm was developed by the author for comparison to the *TIC* algorithm. This implementation was in the method of Steinbach as noted in Section 3.2.

A custom implementation was necessary in order to make use of the document objects outlined in Section 5.2. Furthermore, parameters had to be recorded to evaluate performance of the algorithm. Finally, a data set output for the GUI had to be produced. This set was processed separately from the algorithm implementation, so as not to affect the execution time of the *Bisecting K-Means* algorithm. Additional details of the implementation were as follows.

Documents were represented in a vector space model. Each vector existed in 182-dimension space and had its length normalized to one.

Initial centroids for clusters formed by bisecting the largest existing cluster in the corpus were selected at random from the set of documents in the cluster to be bisected. Documents were assigned to one of the child clusters based on a document's *cosine similarity* to the given two centroids.

The centroids for the two child clusters were then recalculated and the documents redistributed. This process was repeated until the child clusters remained stable.

Support of successful cluster formation was defined at 50. The process was iterated until no cluster remained that could be bisected into two clusters that each contained at least the support level (i.e. 50) documents. The individual cluster bisecting was iterated until either the centroids remained stable and each division contained at least the support number of documents, a successful bisection, or one of the proposed new cluster's size decreased below the level of support, a failed bisection. Clusters were indexed by the cluster's number of formation as well as the three greatest dimensions in the centroids of the cluster.

5.4.4 CLUSTER COHESION

The clusters formed by both the *TIC* algorithm and the *Bisecting K-Means* algorithm were evaluated by determining the cohesion of the individual clusters, the mean, and the standard deviation of the cohesions of all clusters formed.

To determine the cohesion of a given cluster, the centroid for the cluster was initially determined. This was based on the normalized values for the 282 critical terms contained within the documents. For each document in the cluster the Minkowski dissimilarity was then calculated based on the formula:

$$d(i,c) = \sqrt{\left(\sum_{j=1}^{n} \left|x_{ij} - x_{cj}\right|^{2}\right)}$$

Where n is 282, the number of terms, x_{ij} is the value of dimension j in document i, x_{cj} is the value of dimension j in the centroid, and d is the dissimilarity of a document in the cluster from the centroid.

Equation 5. Minkowski metric for dissimilarity

Mean cohesion, c, was then determined by:

$$c = \frac{1}{n}\sum_{1}^{n} d$$

Where n is the number of documents in the cluster

Equation 6. Mean cluster cohesion.

Having calculated the mean cohesion for a cluster, the standard deviation for the cluster cohesion was determined by:

$$s.d. = \sqrt{\frac{1}{N}\sum_{1}^{N}(d-c)^2}$$

Where N is number of documents in a cluster.

Equation 7. Cohesion standard deviation

The value for the mean and standard deviation of all the clusters' cohesion was then calculated for each algorithm along with the maximum (worse case) and minimum (best case) cohesion for a given algorithm. This was calculated in the approach of Equation 6 and 7.

6 RESULTS

6.1 TERM REDUCTION--SELECTION

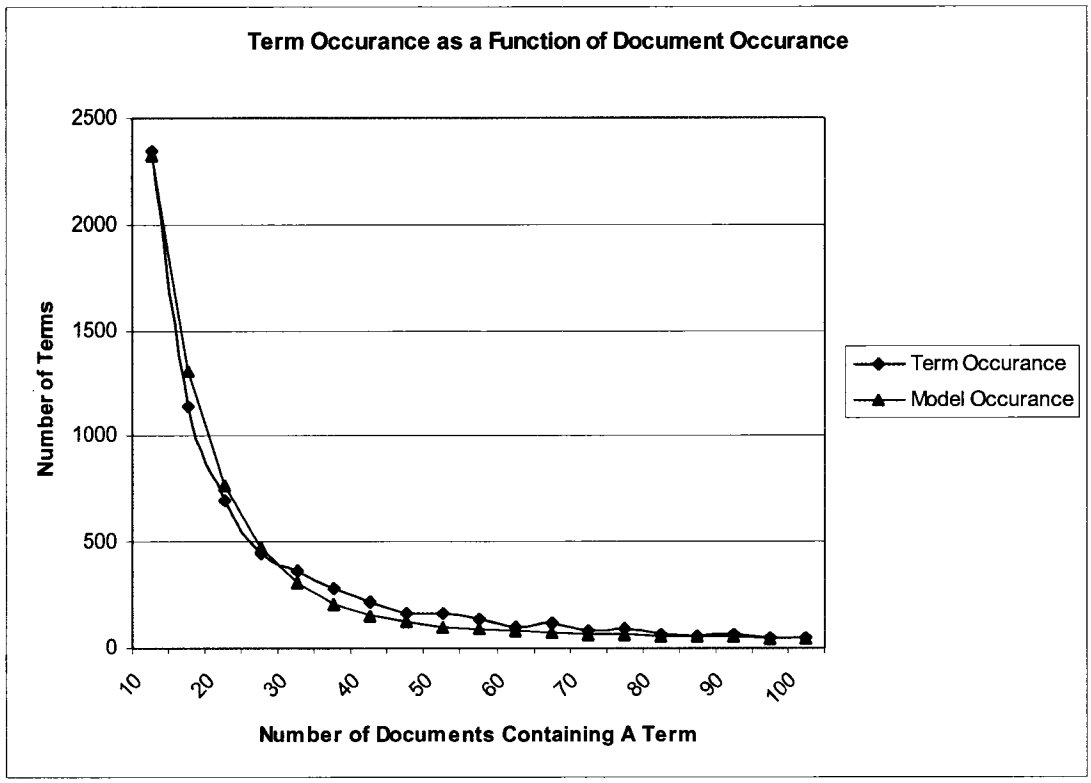
The analysis of both the terms contained in the titles of the Reuter's 21578 News Corpus and the terms contained within the bodies of articles in the Reuter's corpus revealed that the term distribution was inversely exponential. Of 45,691 non-stemmed and non-stopped terms in 19,042 documents, 34,164 terms occur in 5 or fewer documents. Figure 4, demonstrates the distribution of terms with respect to the term's document frequency

over the range of six to 100 documents. The Model Occurrence displayed in Figure 4 is derived from the formula:

$$MF = \frac{45691}{10 * df} + \frac{45691}{7e^{(df/8)}}$$

Where *MF* is the number of Terms, *df* is the number of documents containing the term, and 45691 is the number of unique terms within the corpus.

While the *MF* metric was derived to fit the observed term distribution of Figure 4, the metric is consistent with the data presented in Figure 4 and Figure 2. Initially, there is an inverse exponential drop in the number of documents at a given term frequency. This is represented by the second term in the *MF* metric. At some point, the decrease in number of documents at a given term frequency becomes linear. This is represented by the first term in the *MF* metric. The numerator for the two terms was the number of terms in the corpus. Furthermore, the first term will approximate the second term at a document frequency or 30 (.0333 versus .0336). The implications of this observation are further discussed in Section 7.1.



The number of terms is the sum of terms included in the index value and the proceeding four values. For example, for an x axis value of 10 the y value is the number of terms occurring in 6, 7, 8, 9, and 10 documents.

Figure 4. Term Distribution.

The median value for the document distribution of non-stopped and stemmed terms was 1.2. The Log-normal distribution of terms and the effects of stemming and stop words are further delineated in Table 1.

Doc. Occurrence	Total Terms	Stop Word Terms	Non-Stop Word Terms	Stemmed Terms
>50%	8	8	0	0
25-50%	20	15	5	0
20-25%	10	8	2	0
10-20%	41	7	33	1
Totals	79	38	40	1

Table 1. Term Distribution

Table 1 and Appendix A demonstrates that most terms that are considered to be stop words have document frequencies consistent with the critical term set. Of the 128 stop terms developed by the author for term selection, only 31 had document frequencies of greater than 20 per cent, the maximum range value for the *–idf* value.

The distribution of documents with respect to the number of terms that a document contains was also evaluated. Twenty one documents contained none of the selected critical terms. Over one half of the documents contained eleven critical terms or less. Figure 5 illustrates the distribution of documents with respect to term occurrence.

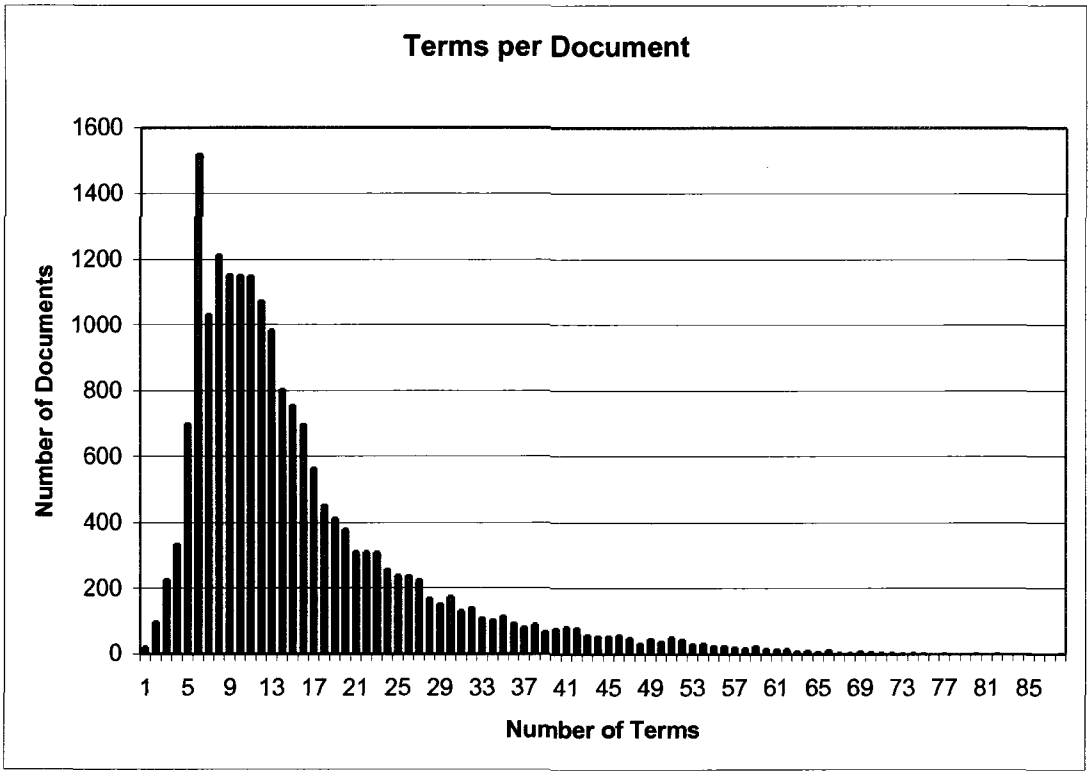


Figure 5. Document distribution with respect to number of critical terms contained within.

Figure 5 in particular demonstrates that for the vast majority of documents, the number of critical terms contained in that document definition is less than 10 per cent of the critical terms. The implications of this are further examined in Section 6.3.

6.2 TERM INTERSECTING CLUSTERING

The execution time for the Term Intersecting Clustering algorithm proposed in this analysis was 75 seconds at a cluster support of 50. An evaluation of execution times versus support is shown in Figure 6.

The number of documents included in the clusters formed was evaluated based on the level of support and the depth of clustering. Furthermore, the number of documents not included in the clusters formed was evaluated.

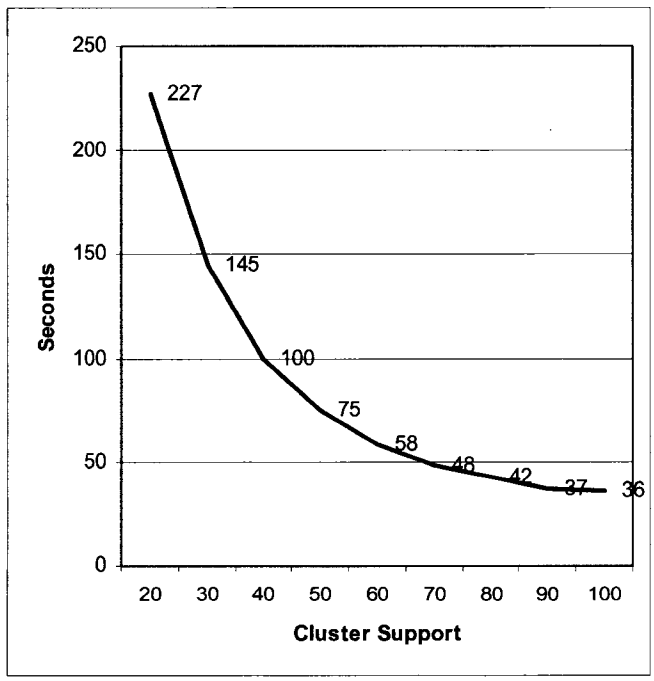


Figure 6. TIC algorithm execution time as a function of cluster support.

Figure 6 demonstrates the intuitive expected execution times of the *TIC* algorithm. As the minimum support value for clusters was increased, the execution time decreased.

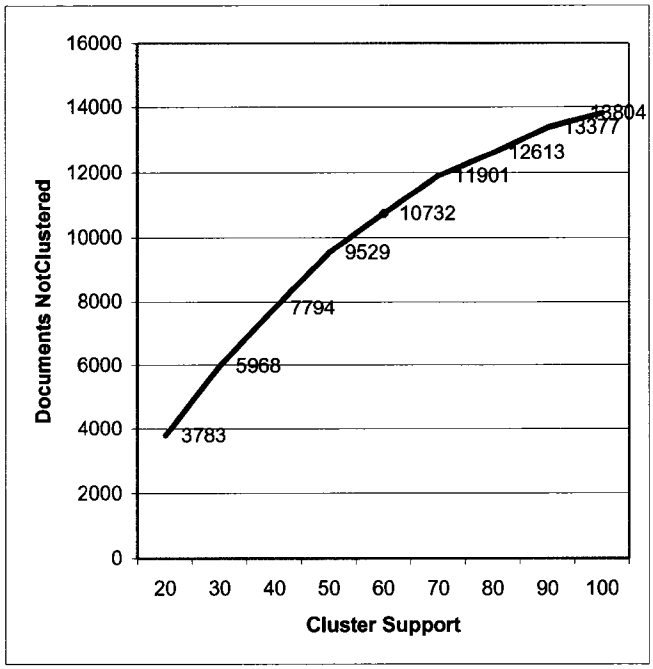


Figure 7. Number of documents not clustered at the three term level versus support for the TIC algorithm.

The rationale for the decrease in execution times in Figure 6 was a consequence of the decrease in the number of clusters formed as the value for cluster support is increased. Figure 7 demonstrates the increase in the number of documents not clustered as the value for cluster support is increased.

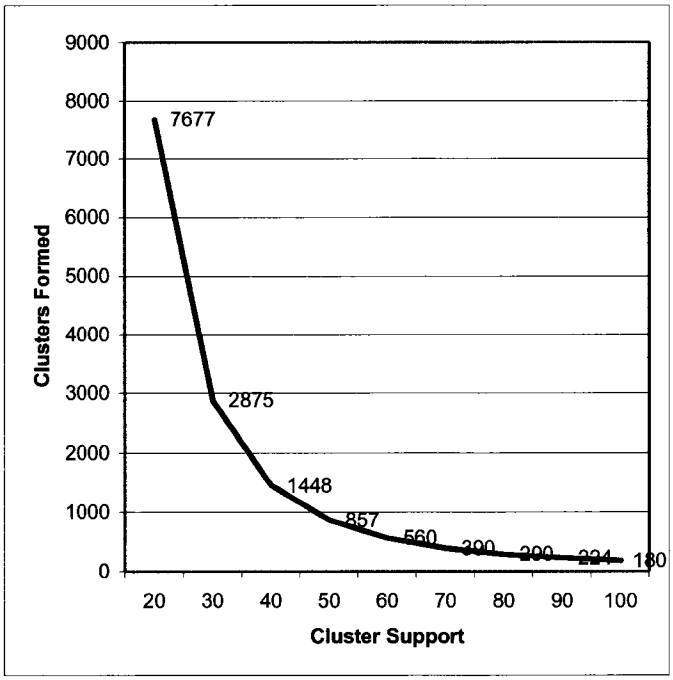


Figure 8. Three term clusters formed as a function of cluster support for TIC algorithm.

In Figure 8, the number of three term clusters formed as a function of the value for cluster support is demonstrated. At a support of approximately 55, the decrease becomes linear in nature. A support level of 50 was selected for the comparison of the *TIC* and *Bisecting K-Means* algorithms, as system memory prohibited the creating of a GUI for the *Bisecting K-Means* algorithm below this level. A functional GUI could be created for the *TIC* algorithm to a minimum support level of 20.

The cohesion of the clusters formed and the standard deviation of that cohesion was calculated. The final results of clustering to a depth of three terms with a cluster support of 50 are listed in Table 2.

Execution time: 75 seconds. Documents that fail to form a cluster: 21 (<.2%) Documents that fail to form a two term cluster: 384 (2.0%). Documents that fail to form a three term cluster: 9529 (50%). Two term clusters formed: 1025. Three term clusters formed: 857. Mean three term cluster cohesion: 0.7211. Three term cluster cohesion standard deviation: 0.1632. Maximum three term cluster cohesion: 0.9917. Minimum three term cluster cohesion: 0.1315.

Table 2. Results of TIC algorithm with cluster support of 50.

Table 2 demonstrates that 98% of the documents formed either two or three term clusters. The remaining 2% of documents (384) are outlying values. This was felt to be reasonable and consistent with the corpus.

When Table 2 is contrasted to Table 4, the results of the *Bisecting K-Means* algorithm, it is observed that the cohesion for the *TIC* algorithm is inferior to the former. The significance of the difference in cohesion is discussed in Section 7.2.3.

6.2.1 *TIC* GUI

A GUI was then created to allow manual inspection of the clusters formed. For the *TIC* algorithm, the clusters were indexed by the critical terms used to create the clusters. The GUI created a tree with the parent nodes being the number of terms in a cluster, the index terms of the cluster, and the title of the articles in the cluster. The child nodes were the text of the body of the article. The structure of the GUI and screen examples are shown in Figures 9 through 13.

Figure 9. Tree Structure of TIC GUI

In Figure 9 the tree structure of the GUI is demonstrated to a given leaf node (Document 668). The leaf nodes in the GUI contain the actual text of the body of a given document to allow recall and evaluation of the text of a document.

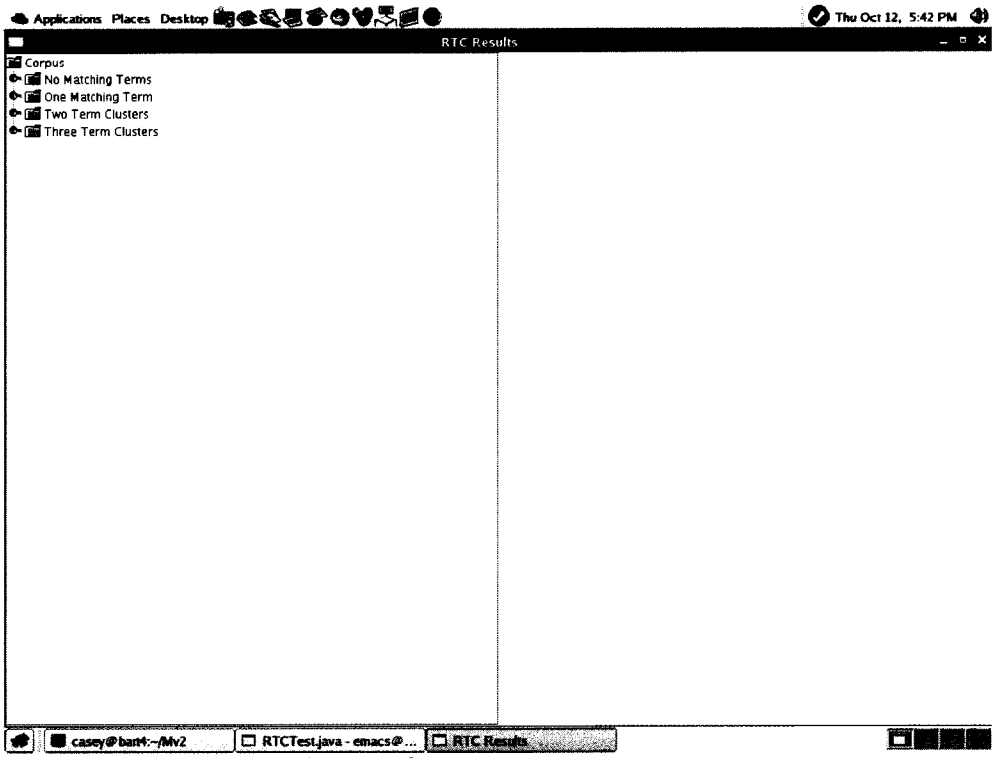


Figure 10. Root screen image of TIC GUI.

Figure 10 is a screen shot of the actual GUI's initial display. The four nodes consist of documents that contain no critical terms, documents that contain only one critical term, and clusters that contain documents with two and three critical terms.

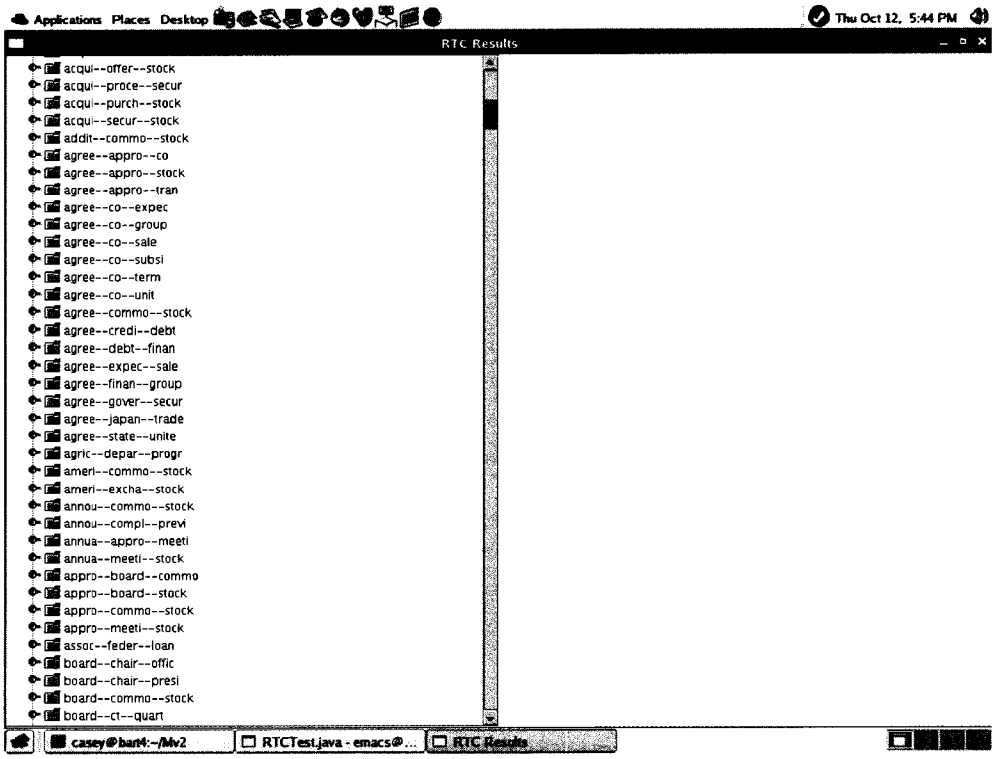


Figure 11. Example of three term clusters.

Figure 11 demonstrates the child nodes of the three term node. The names of the child nodes are the indexing terms that created the given cluster.

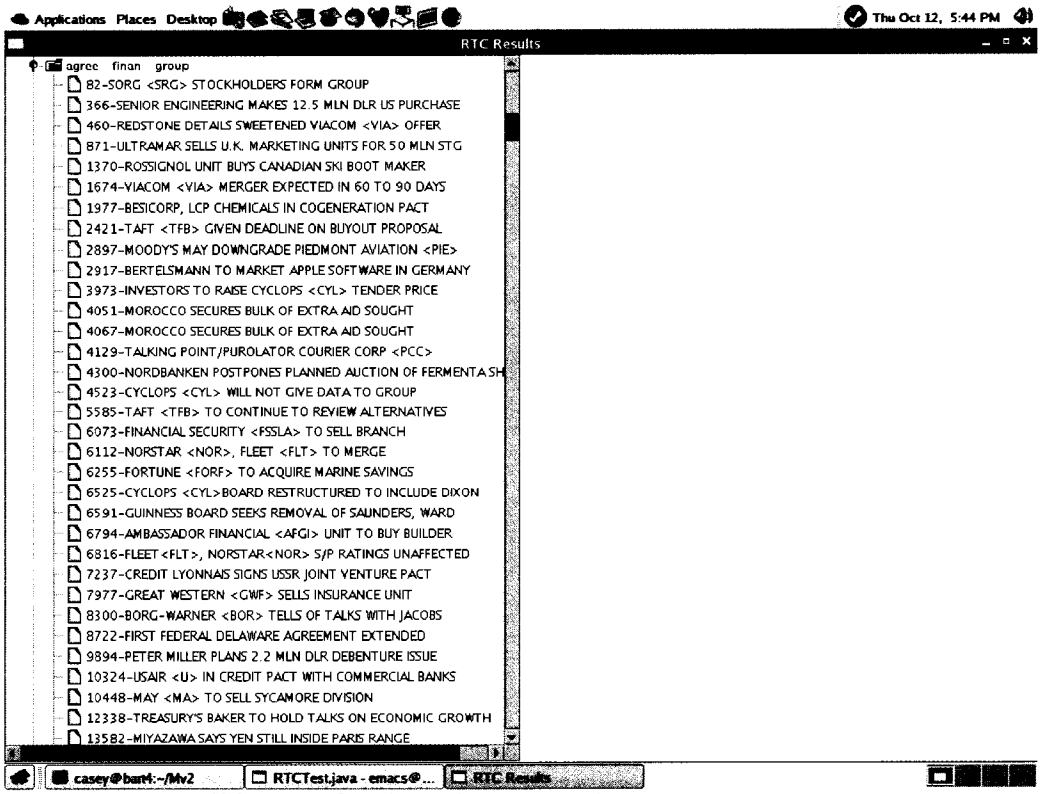


Figure 12. Detail image of a given three term cluster

Figure 12 displays the document object number and document title of the documents contained in a given indexed cluster. By double clicking on a given title, Figure 13 is displayed, with the text of the document body displayed in the right side frame.

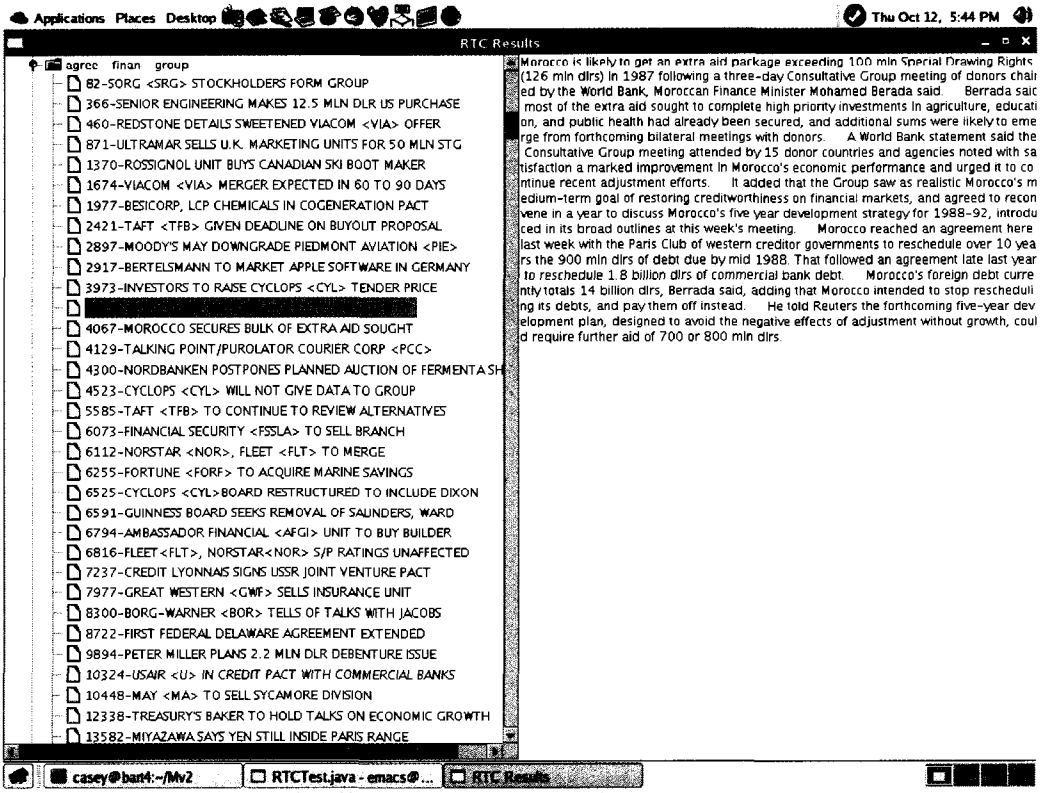


Figure 13. Document detail from a three term cluster.

6.3 BISECTING K-MEANS CLUSTERING

In the author's implementation of the *Bisecting K-Means* algorithm, the execution time varied from run to run at any given support level. The minimum execution time recorded was 31 seconds. This was recorded at a variety of support levels from 20 to 100 documents per cluster. The maximum execution time was 43 seconds recorded for a support of 60 documents per cluster.

The number of clusters formed as a function of cluster support was found to have no significant difference in the support range of 20 to 100 documents. The total number of cluster attempts trend appeared to be a function of the inverse of the cluster support value over the same range. These values are demonstrated in Figure 14.

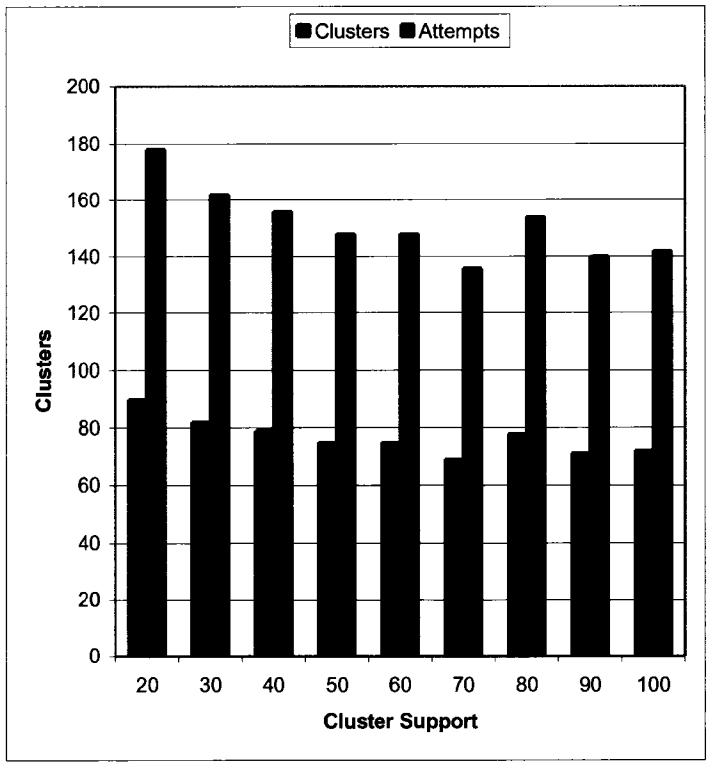


Figure 14. Clusters formed as a function of the cluster support (*Bisecting K-Means* algorithm).

Unlike the *TIC* algorithm, the *Bisecting K-Means* algorithm did not demonstrate a relation between the number of clusters formed and the cluster support value. The author's implementation of the *Bisecting K-Means* algorithm was not fuzzy (i.e. a document could only exist in one cluster).

By the nature of the *Bisecting K-Means* algorithm, all documents were included in a cluster. The mean cohesion for all clusters formed was 0.197. The range of cluster cohesions as a function of cluster support value is shown in Figure 15.

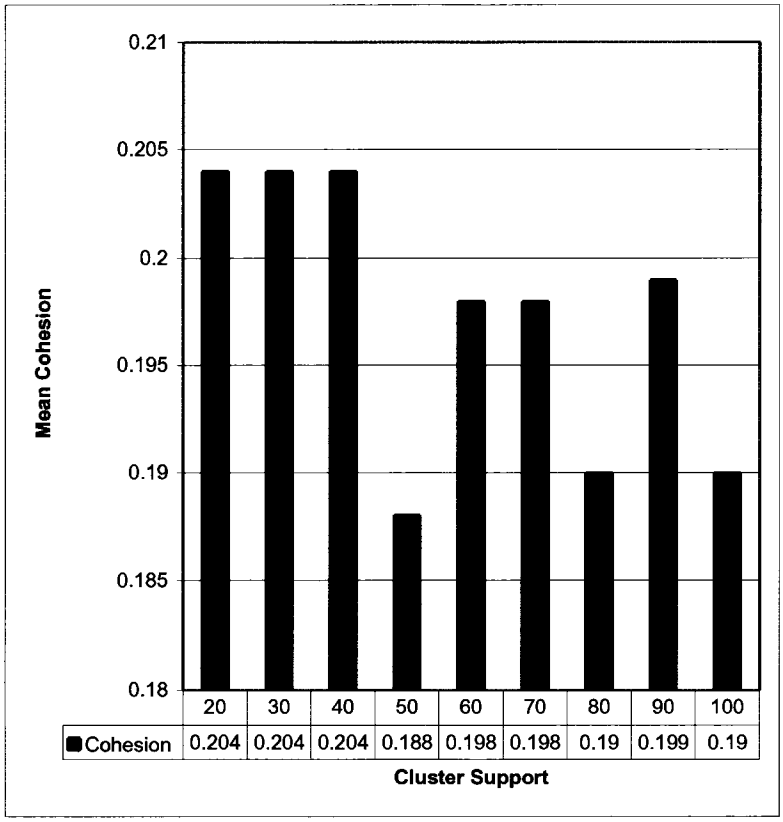


Figure 15. Mean Cluster Cohesion as a function of Cluster Support.

Figure 15 demonstrates a trend for smaller values of cohesion (tighter clusters) as the support value is increased over the range of 20 to 100 documents. The amount of variation makes this trend non-significant.

The centroids of the clusters formed were evaluated for the number of dimensions that had attributes greater than zero. The maximum number of dimensions was 182. Table 3 contrasts the properties of the centroid vectors with those of the document vectors.

	K-Means Centroids	Document Objects
Mean	163.91	14.8
Median	177.50	11.5
Mode	182.00	7.0

Table 3. Comparison of Centroids versus document dimensions.

In Table 3, the significant difference in the properties of the centroid utilized for the *Bisecting K-Means* algorithm and the individual documents is contrasted. While the documents are defined by a mean value of 11.5 non-zero dimensions, the cluster centroids are defined by a mean value of 177.5 non-zero dimensions. In the author's opinion, this significant difference creates the discrepancy between the level of cohesion and the actual value of the cluster as discussed in Section 7.2.3. The distribution of the dimensions of cluster centroids is further presented in Figure 16.

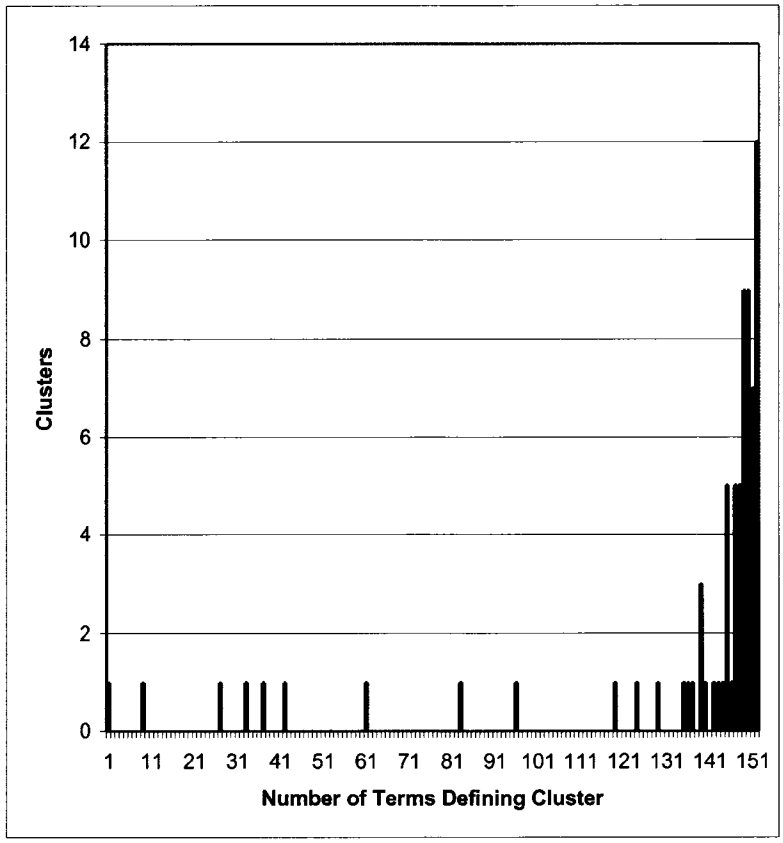


Figure 16. Cluster distribution as a function of critical terms contained within the cluster centroid.

A summary of the *Bisecting K-Means* algorithm results for a support level of 50 is presented in Table 4.

Execution time: 31 seconds. Clusters formed: 75 Mean cluster cohesion: 0.188 Cohesion standard deviation: $6.58*10^{-3}$ Maximum cluster cohesion value: $6.18*10^{-1}$ Minimum cluster cohesion value: $1.02*10^{-1}$

Table 4. Summary of Bisecting K-Means Clustering .

Table 4 reports on one result of the execution of the *Bisecting K-Means* algorithm's program. As noted above, the execution time of 31 seconds was the best time recorded for the algorithm. Furthermore, the mean cluster cohesion was also the best value recorded for the *Bisecting K-Means* algorithm during the evaluation. A maximum execution time of 43 seconds was recorded. The number of clusters formed and the cohesion values varied from one run to another. This was a result of the random selection of documents for the initial centroids in cluster formation. In general, the values for cohesion were smaller (favorable) for the *Bisecting K-Means* algorithm than the *TIC* algorithm.

6.3.1 BISECTING K-MEANS GUI

A GUI was created to allow evaluation of the clusters formed by the *Bisecting K-Means* algorithm. The clusters were initially indexed by the order that the cluster were formed. As this numeric value was of little intuitive value in evaluating the clusters, the clusters were also indexed by the three terms that had the largest value in the centroids of the cluster.

The resulting GUI was similar to the GUI for the Term Intersection Clustering algorithm as demonstrated in Figure 17.

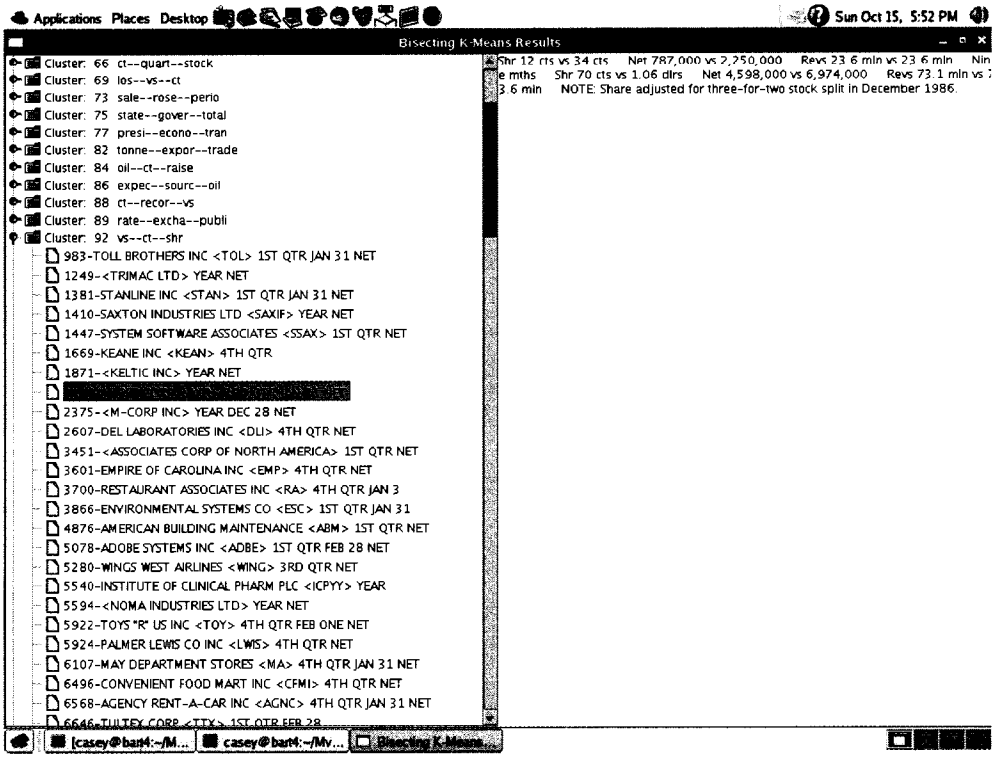


Figure 17. GUI of Bisecting K-Means results.

The results of a manual inspection of the clusters in the GUI revealed that the content of the documents in the clusters did not appear as closely related as the content for the three term clusters of the *TIC* algorithm. This is discussed in Section 7.2.3.

Another observation was that the three index terms of a given cluster did not have to be present in every document in that cluster. This is reviewed further in Section 7.2.3.

7. DISCUSSION

The focus of this thesis was the comparison of the *TIC* and the *Bisecting K-Means* algorithms. An evaluation of the distribution of terms within the corpus, as reported in Section 6.1, also deserved further comment, based on the findings.

7.1 TERM REDUCTION & SELECTION

An implicit concept in the utilization of the *tf-idf* metric is that there exists a Gaussian distribution of terms within the documents of a corpus. For the classic Gaussian distribution, the mean, median, and mode for the distribution will all be equal. The distribution is defined by the mean and the standard deviation of the distribution. The statistics of the Gaussian distribution have been well studied. A two standard deviation from the mean is a frequent value to reject values from a sampling set.

A distribution that resembles the bell curve distribution of a Gaussian distribution, but that has mean, median, and mode values that are not equal has frequently been referred to as a non-Gaussian distribution. A function that describes the results of sampling objects (such as terms) rarely is Gaussian in nature as the function is not continuous at zero. Despite the non-Gaussian nature of samplings, the statistics of the Gaussian distribution are frequently utilized successfully.

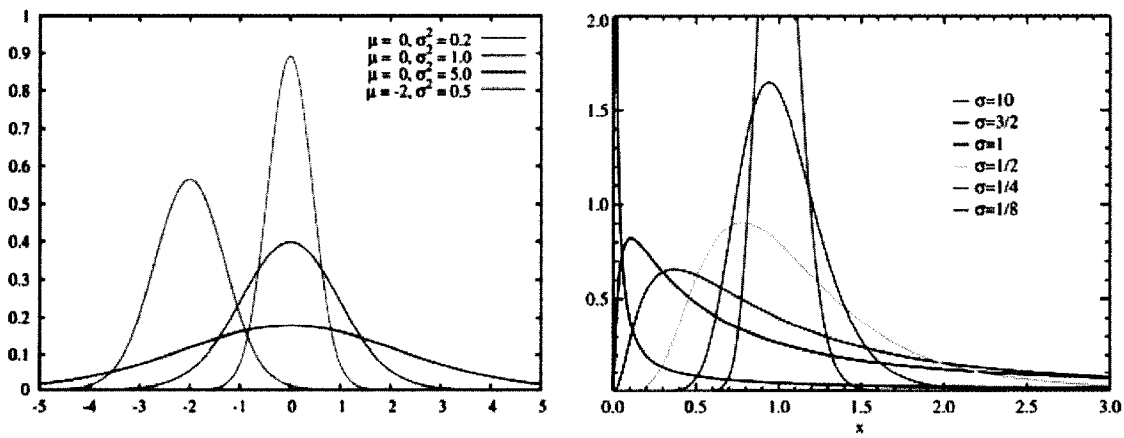


Figure 18. Gaussian distribution versus Log-normal distribution (Representative graphs--Wikipedia).

The distribution of terms both before and after utilization of stop words and stemming can best be defined as a Log-normal distribution for the Reuter's News corpus. As this study did not evaluate other document corpuses, no comment can be made whether this distribution is the normal occurrence for text documents, or a unique occurrence.

If term distribution is generally of a Log-normal variant for document corpuses, further study would be warranted into the development of a metric that employed the statistics of Log-normal distributions. This would be of value in eliminating the somewhat random selection of *-idf* values utilized in term selection as performed in this study as well as in Moch's study.

7.2 TIC VERSUS BISECTING K-MEANS CLUSTERING

The *TIC* and the *Bisecting K-Means* algorithm were compared on three levels. Those levels were the execution time, the reproducibility of the results, and the quality of the clusters. Furthermore, the concept of cluster quality was evaluated.

7.2.1 EXECUTION TIME

As noted in Table 2 and Table 4, the *Bisecting K-Means* algorithm produced faster execution times than the *TIC* algorithm, although the execution times were of the same order of magnitude. This difference ranges from 32 to 44 seconds in favor of the *Bisecting K-Means* algorithm. The difference in the execution times may be more a function of the implementation than of the algorithms, as explained below.

The time to parse and prepare the corpus into the Java document objects was not included in the execution time as the Java documents were employed by both algorithms. Included in the document object was the Euclidian length of the given document object. This value was necessary to calculate the cosine similarity of documents. The length value was required in the *Bisecting K-Means* algorithm, as well as the Minkowski dissimilarly that was employed in calculating cluster cohesion. The length value had a limited role in the *TIC* algorithm, and was utilized as it was available within the document object.

Another factor in the execution time difference was the manner in which the *Bisecting K-Means* algorithm was implemented. For a corpus of 19042 documents and utilizing 182 terms, the entire set of document vectors could be contained in a Java array defined as *double [19042] [182]*. This was utilized to minimize the execution time for comparison of the *Bisecting K-Means* algorithm to the *TIC* algorithm. The effect of employing Java TreeSet objects versus an array of java primitives was not evaluated.

The *TIC* algorithm made use of the intersection of Java TreeSet objects to perform clustering. The overhead of manipulating Java objects versus Java primitives was not determined. The TreeSet objects were utilized to allow the clustering of larger corpuses than the Reuter's Corpus. The array employed for the *Bisecting K-Means* algorithm was limited by system memory in how large a corpus can be clustered.

7.2.2 REPRODUCIBILITY

The *TIC* algorithm will always produce the same set of clusters for a give static corpus. This reproducibility may not necessarily be the case for the *Bisecting K-Means* algorithm.

In the implementation presented, the initial centroids for the clusters formed in the execution of the *Bisecting K-Means* algorithm, were selected at random. This produced different cluster results with every execution of the algorithm. While the results were different, the cohesion and cohesion standard deviation were similar.

To introduce recall and to evaluate the clusters formed, an additional method was created to process the cluster results into a GUI. This method was not included in the *Bisecting K-Means* execution time although the program is required to recall the identical results of execution of the *Bisecting K-Means* algorithm.

7.2.3 CLUSTER QUALITY

The primary purpose of any clustering algorithm is information retrieval. The consequence of this objective is to produce clusters of a high quality and reproducibility. The question is what defines a high quality cluster?

In this evaluation, the cohesion of clusters was contrasted between the *Bisecting K-Means* algorithm and the *TIC* algorithm. As demonstrated in Table 2 and Table 4, the *Bisecting K-Means* algorithm produced superior cluster cohesion versus the cohesion of the clusters formed by the *TIC* algorithm. The question that remained was which algorithm, if either, produced the superior quality clusters.

The standard *Bisecting K-Means* algorithm will cluster every document in the corpus. There is no provision for handling outlying documents. This was problematic for the 21

documents in the corpus that contained no significant terms. Those documents were included in the cluster(s) formed by the *Bisecting K-Means* algorithm. In the *TIC* algorithm, those documents were segregated into their own cluster.

Another challenge to the *Bisecting K-Means* algorithm and the utilization of the *cosine similarity* metric is the significant difference in the number of unique terms in a given document object (Figure 5) and the number of unique terms in the cluster centroids (Figure 16).

The centroids calculated were a function of the union of all documents in a cluster. The centroids computed for the Reuter's 21578 News Corpus had a median value for non-zero attributes of 177.5 out of 182 possible attributes (Table 3). This is contrasted to a median value for non zero attributes in the individual documents of 11.5 out of a possible 182 attributes.

The significant difference in the non-zero dimensions of the centroids of the clusters and the documents that the clusters contain is further exemplified by the results of an attempt to form tour term clusters utilizing the *TIC* algorithm. When this was evaluated, with a support level of 20, 66 per cent of the documents did not cluster (i.e. 66% of the documents did not have four terms in common with 19 other documents).

Elements were compared to the centroids of clusters in the *Bisecting K-Means* algorithm utilizing the *cosine similarity* metric, not directly to each other. The result of this metric

was that a cluster may contain documents that had few or no non-zero attributes in common as noted above.

The GUI that was created to evaluate the results of the *Bisecting K-Means* algorithm was indexed by the three largest non-zero attributes in the centroid of the cluster in question. Documents within the cluster had no requirement to contain non-zero values for any or all of the three indexed terms.

A potential limitation of the *TIC* algorithm was the exclusion of documents based on the lack of common terms linking documents. Table 2 demonstrates the number of documents that failed to cluster at a given term level. Only 50 percent (9513 documents out of 19042 documents) were included in clusters at the three term level when cluster support was set to 50. An advantage of the *TIC* algorithm was that it properly identified documents that contained no critical terms. Furthermore, 98 per cent of the documents were included in either a two or three term clusters.

The calculation of F-values for evaluation of cluster quality would have been beneficial. The original goal was to utilize the attributes recorded in the topic and location tags of the original SGML documents to perform this task. Unfortunately, the attributes were of little value when a sampling of the documents was considered.

The final evaluation of the quality of the clusters created was a manual sampling of those clusters via a GUI created for the *Bisecting K-Means* output and the *TIC* output. There was not a complete overlap of the index terms for the *Bisecting K-Means* algorithm and

the *TIC* algorithm. An index that did exist in both was "excha, offer, state". In the *Bisecting K-Means* index the ranking of the terms was "offer, state, excha". As noted above, this was determined by the magnitude of these dimensions in the centroid vector.

Review of the *TIC* algorithm results revealed that the documents contained within the cluster, 105, all contained reference to the offering of financial instruments (i.e. stocks, bonds, notes). Some of the documents also contained information on earnings reports and dividends in addition to the offering of the financial instruments.

The review of the *Bisecting K-Means* results for the same index produced 247 documents in the cluster formed. There was only one cluster formed with the above three term index, although multiple clusters with the same index terms were possible with the *Bisecting K-Means* algorithm. A review of the cluster revealed that only 18 percent concerned the offering of financial instruments. Other topics contained in the cluster included; an earth quake in Central America, the fall of an Italian government, the indictment of a former White House aide, the raising of the national speed limit, and numerous non United States political activities.

The *TIC* algorithm results were then evaluated at the two term level. Unfortunately, no combination of "excha, offer, state" was created to evaluate. The cluster indexed on "excha, forei" was selected.

This cluster contained 41 documents. These documents did not cluster at the three term level. Of these 41 documents, 22 concerned the economic reserves of non United States

countries (54 percent). Six of the articles concerned a foreign exchange scandal that occurred involving Volkswagen AG (15 percent).

As only 384 documents were not contained in a two or three term clusters. This set of 384 documents formed two clusters in the output of the *TIC* algorithm, one cluster of 21 documents that contained no critical terms, and one cluster of 363 documents that contained only one critical term.

This example sampling was consistent with the observations of other clusters formed by the *TIC* algorithm and the *Bisecting K-Means* algorithm.

8. CONCLUSIONS

The *Bisecting K-Means* algorithm is considered the *Gold Standard* by which clustering algorithms are judged. The reason is reasonable execution time combined with superior cluster cohesion.

In this study, the *Bisecting K-Means* algorithm demonstrated the above attributes. Execution time was better, although only slightly than the *TIC* algorithm, and cohesion was more than three times better for the *Bisecting K-Means* algorithm. The limitations of the *Bisecting K-Means* algorithm were only evident when the actual clusters formed were evaluated.

As noted above, the *Bisecting K-Means* algorithm does not lend itsself well to indexing of the clusters that are formed. This study proposed and utilized a workable solution.

The execution time to fix and index the results of the *Bisecting K-Means* algorithm was not included in the execution time of the *Bisecting K-Means* algorithm. If this time had been included, the execution time of the *TIC* would have been less than the execution time of the *Bisecting K-Means* algorithm.

Furthermore, the *Bisecting K-Means* algorithm does not produce consistent results with respect to the contents of the clusters formed. This was secondary to the random selection of documents as the initial centroids for the clusters formed by the bisection of their parent cluster. As the selection is random, the bisected clusters formed may consist of different documents on each execution of the algorithm. This study also proposed and adopted a workable solution in which the results were fixated into a GUI by a secondary program.

The significant limitation of the *Bisecting K-Means* algorithm was with the content of the clusters formed. Despite better cohesion, a selective, subjective, review of the contents of clusters revealed inferior cluster quality at both the three term and two term cluster levels of the *TIC* algorithm.

The *Bisecting K-Means* algorithm has no provision to deal with outlying documents in the clustering process. On the contrary, the *TIC* algorithm excluded 384 documents with zero or only one critical term. In the implementation of the *Bisecting K-Means* algorithm, these 384 documents may be randomly selected as an initial centroid for a new cluster that is formed by the bisecting metric. This may explain the inferior content of the clusters formed by the *Bisecting K-Means* algorithm versus the *TIC* algorithm.

While the *TIC* algorithm was inferior to the *Bisecting K-Means* algorithm with respect to cohesion, this study demonstrated far superior results in the content of the clusters formed. If the documents in the Reuter's 21578 News Corpus were correctly classified, the expected F-values for the *TIC* algorithm would be superior to the findings for the *Bisecting K-Means* algorithm. Furthermore, the indexing of the clusters formed by the *TIC* algorithm gave insight into the content of the given cluster. The superior content, ease of indexing of the *TIC* algorithm, and execution times indicate that the *TIC* algorithm is superior to the *Bisecting K-Means* algorithm for information retrieval.

9. FUTURE STUDY

The algorithms reviewed in this study all made use of the bag of words concept to represent the documents contained within their corpus. Critical to the process is selecting the correct terms to represent documents. The term distribution in the Reuter's 21578 News Corpus illustrated the weakness in the current commonly employed metric that was proposed by Salton [19] over 20 years ago.

A fundamental concept of the *tf-idf* metric is a Gaussian distribution of terms. For the Reuter's News Corpus, that is not the case. Therefore, the author proposes to evaluate additional corpuses of text documents. If the Reuter's case is unique, then the use of the Reuter's 21578 News Corpus to evaluate clustering algorithms would be called into question. If the Reuter's case is not unique, then consideration of a new metric for term selection is warranted. The goal would be to develop a metric that would eliminate the arbitrary assignment of limits on documents –*idf*, when performing term selection.

Another area that requires further work is the classification of the Reuter's 21578 News Corpus documents. While there are tags in the SGML files for document topics, location, and people, the values contained are not consistent. Lewis [13] on his web site discusses further work being done on the corpus. Hopefully, this area will be addressed.

A suggestion would be for one author to evaluate a small random sampling of the corpus and to develop a set of non-exclusive topics for article classification. The author could then supervise others in the assignment of topics to all the articles within the corpus.

A final area that warrants further evaluation is the validity of the vector space model for document abstraction. This study demonstrated excellent cohesion of the clusters formed by the *Bisecting K-Means* algorithm. When the clusters created by the *Bisecting K-Means* algorithm were examined, there exists a question to the actual quality of the clusters. In this author opinion, the probable cause is the significant difference in the number of dimensions that define individual documents versus the number of dimensions that define cluster centroids as demonstrated in Table 3.

REFERENCES

[1] Agarwal, R., Aggarwal, C.,Prasad, V. "A Tree Projection Algorithm For Generation of Frequent Itemsets", Journal of Parallel and Distributed Computing, 2001.

[2] Baeza-Yates, R., Ribeiro-Neto, B, *Modern Information Retrieval.* ACM Press, 1999.

[3] Bartman, C., Alsabbagh, J. "Reverse Tree Clustering", DMIN06, pp364-367. Las Vegas, N.V. 2006

[4] Beil, F., Ester, M., Xu, X., "Frequent Term-Based Text Clustering", SIGKDD 02, pp 436-442, Edmonton, Alberta, Canada, 2002.

[5] Chen, H., Houston, A., Sewell, R., Schatz, B.,"Internet browsing and searching: User evaluations of category map and concept space techniques", Journal of the American Society for Information Science, vol 49, issue 7, pp 582-603, 1999.

[6] Dong, X., Halevy, A., Madhqvan, J., Nemes, E, Zhang, J., "Similarity Search for Web Services", Proceedings of the 30th VLDB Conference, pp372-383., Toronto, Canada, 2004

[7] Fasulo, D. "An Analysis of Recent Work on Clustering Algorithms", Department of Computer Science & Engineering, University of Washington, 1999.

[8] Frakes, W, Baeza-Yates, R. *Information Retrieval: Data Structures & Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1992.

[9] Jain, A., Dubes, R., *Algorithms for Clustering Data.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

[10] Jain, A., Murty, M., Flynn, P. "Data Clustering: A Review", ACM Computing Surveys, Vol. 31, No. 3, pp 264-323. September 1999.

[11] Karypis, G., Eui-Hong, H., Kumar, V. "CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling", Computer,Volume 32, Issue 8, pp 68 – 75, Aug 1999.

[12] Larsen, B., Aone, C., "Fast and Effective Text Mining Using Linear-time Document Clustering", KDD-99 pp. 16-22, San Diego, CA, 1999.

[13] Lewis, D., Available: http://www.daviddlewis.com/resources/testcollections/reuters21578/. Last visited on November 5, 2006.

[14] Mock, K, "A Comparison of Three Document Clustering Algorithms: TreeCluster, Word Intersection GQF, and Word Intersection Hierarchial Agglomerative Clustering", Intel Technical Report, September 1998.

[15] OCLC. *www.oclc.org.* Dublin, Ohio. 2006.

[16] Osinski,S., Weiss, D., "A Concept-Driven Algorithm for Clustering Search Results", *IEEE Intelligent Systems*, p. 48, May/June 2005.

[17] Porter, M.F. "An algorithm for suffix stripping", Program, 14 no. 3, pp 130-137, July 1980.

[18] Salton, G., "Automatic Text Indexing Using Complex Identifiers", ACM Conference on Document Processing Systems, pp135-136, 245-246, Santa Fe, N.M., 1988.

[19] Salton, G. *Automatic Text Processing.* Addison-Wesley, 1989.

[20] Steinbach, M., Karypis, G., Kumar, V., "A Comparison of Document Clustering Techniques," University of Minnesota, Technical Report #00-034 (2000). http://www.cs.umn.edu/tech_reports/.

[21] Voorhees, E. "Implementing agglomerative hierarchic clustering algorithms for use in document retrieval". Information Processing and Management, 22(6), pp 465-467.

[22] Weiss, S., Indurkhyz, N., Zhang, T., Damerau, F., *Text Mining, Predictive Methods for Analyzing Unstructured Information.* Springer Science+Business Media, New York, N.Y. 2005.

[23] Willett, P. "Recent trends in hierarchic document retrieval: A critical review". Information Processing and Management, 24, pp 577-597.

[24] Zamir, O., Etzioni, O., Madani, O., Karp, R., "Fast and Intuitive Clustering of Web Documents", Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, pp 287-290, 1997.

[25] Zamir, O., Etzioni, O., "Word Document Clustering: A Feasibility Demonstration", *SIGIR 98*, pp 46-54, Melbourne, Australia. 1998.

Appendix A

Stop Words

able	eight	more	the
about	end*	most	their
after	every (beginning)	need (beginning)	there
again	exist (beginning)	next	they
all	febru	nine	thi
also	first	no	thing (beginning)
an	five	not	to
and	for	now	today
any	four	of	toget (beginning)
april	from	on	thoug (beginning)
are	fully (beginning)	one	three
as	furth	only	throu
at	going (beginning)	open (beginning)	try (beginning)
away (beginning)	ha	or	two
be (beginning)	had	other	us (beginning)
being	have	out	wa
best	he	over	want (beginning)
bigge (beginning)	high (beginning)	part per	we
billi	if	possi	went (beginning)
bln	impor	said	well
both	in	same	were
but	into	saw (beginning)	when
by	is	seen (beginning)	which
came (beginning)	it	seven	while
can	janua	since	who
could	june	six	will
did	know (beginning)	some	with
done (beginning)	like (beginning)	st	work (beginning)
dont (beginning)	made	such	would
down	make (beginning)	take	you (beginning)
each	march	th	
earli	may	than	
effect		that	

(beginning) notes all terms beginning with the given expression.

-

Appendix A

Custom Stems

Term beginning with the following Strings, return those Strings.

call	mine	stop
crop	own	vote
end*	run	weak
grow	seek	

* The term end was first considered a stop term, and then any term beginning with 'end' was stemmed to end.

-

Appendix B

Stemmed Term List & Document Frequency

accor--1084
accou--1059
acqui--2211
added--1790
addit--1130
agree--3644
agric--973
allow--992
ameri--1573
amoun--1037
analy--1153
annou--1626
annua--1183
appro--2158
asset--999
assoc--1013
board--1551
bond--1188
busin--1673
buy--1017
call--1421
canad--1184
capit--1528
cash--1138
centr--1143
chair--1412
chang--1381
close--1385
co--3008
comme--1984
commi--2612
commo--2134
commu--1017
compe--1028
compl--1697
conce--1302
consi--1402
consu--1038
conti--2155
contr--2820
conve--1163
cost--1262
count--2407
credi--1760
ct--3586
curre--3232
cut--1220
day--1511
debt--1644
decli--1490
depar--1056
devel--1899
direc--1359
dolla--1352
dome--1007
due--1938
durin--1035
econo--2573
elect--1298
end--1658
estim--1236
europ--1230
excha--2455
expec--3514
expor--2132
feder--1367
figur--995
finan--3513
firm--1359
follo--1354
forei--1636
franc--1142
fund--1275
futur--1086
gain--1093
gener--1817
gover--2649
group--2198
grow--1605
hi--1110
house--1005
howev--1037
impro--994
inclu--2974
incre--3191
indu--2805
inve--3103
issue--2480
japan--2014
large--1608
late--1013
level--1373
loan--1487
los--1627
lower--1149
ltd--1692
major--1698
manag--2335
meeti--1550
membe--1052
mini--2209
month--3249
natio--2084
negot--1013
net--3241
note--2160
offer--2894
offic--3430
oil--1455
opera--2801
own--1752
parti--1127
pay--1442
payme--1228
perio--1365
plan--1993
point--1132
polic--1169
pres--1063
presi--1928
previ--1584
prior--1094
priva--1092
proce--1419
profi--1997
progr--1508
propo--1906
provi--1972
publi--1519
purch--1416
quart--1604
raise--1019
rate--2569
recei--1384
recen--1409
recor--1819
reduc--1830
remai--1427
repor--2989
repre--1015
requi--1031
reser--1211
resul--1733
reute--988
rev--1485
rise--1230
rose--1197
sale--3036
secur--2210
servi--1600
set--1196
short--1221
shoul--1262
shr--3018
sourc--1223
south--1028
speci--1100
spoke--1594
start--1168
state--3702
stock--3522
subsi--1527
suppl--1101
suppo--1077
syste--1765
tax--1037
term--1513
time--1683
told--2393
tonne--1093
total--2393
trade--2984
tradi--1401
tran--1715
unit--1673
unite--1092
up--3049
value--1098
vs--3075
week--2294
world--1436
yeste--1110

Appendix C

The Following is a Complete Reproduction of the Reverse Tree Clustering Paper Presented at DMIN06 Las Vegas, NV, June, 2006

Reverse Tree Clustering

Casey R. Bartman, and Jamal Alsabbagh, Grand Valley State University

Abstract—**Common document clustering algorithms utilize models that either divide a corpus into smaller clusters or gather individual documents into clusters. Hierarchical Agglomerative Clustering, a common gathering algorithm runs in $O(n^2)$ to $O(n^3)$ time, depending on the linkage of documents. In contrast, Bisecting K-Means Clustering has been shown to run in linear time with respect to the number of documents to cluster, although other factors significantly affect run time. We propose a clustering algorithm bases on an inverted-index matrix of terms and an inverted term tree model.**

I. INTRODUCTION

As the number of documents available from the web and other sources increase, there has developed a need for robust document clustering algorithms. Most clustering algorithms represent documents as vectors in *n*-dimensional space, where *n* is a set of terms that occur in the documents. Documents are then clustered based on the similarity of their vector representation [1]. Some shortcomings of this approach have been well documented [2]:

1. Many dimensions for a given document are null. This results in a large number of multiplications that have no affect when utilizing methods such as cosine similarity. Operational overhead is therefore introduced for dimensions that are only significant when comparing a limited number of documents in the corpus.
2. *K-means* variations may require fore knowledge of the number of clusters that should be created. This value may be unknown, especially in unsupervised clustering.
3. The evaluation of outlying documents. Are they secondary to a need for an additional cluster per two above? Do they represent a need for larger variation in the clusters?

This paper presents an implementation of an algorithm based upon Mock's Tree Clustering technique [3], whereby the vector model is not used. Instead, documents are associated with terms and clustering is performed based upon them intersection. The advantage of this approach is that the overhead due to null dimensions (in the vector model) is eliminated. This work also implements Mock's proposal for generating overlapping clusters. In addition, cluster generation was accomplished by utilizing an inverted-index matrix as described by Salton [2].

This preliminary study demonstrated a robust method for classifying short text documents. Our goal was to develop clusters of reasonable significance to the end user, rather than to seek tight cohesion among clusters. Ongoing evaluation is underway to compare Tree Clustering with Bisecting K-Means Clustering regarding actual run times against the same corpus and the quality of the clusters formed.

II. THE TREE CLUSTER MODEL

The basis of Mock's Tree Cluster technique is an Apriori algorithm that used an item set lattice to a depth of three terms. Candidate clusters were generated by brute force utilizing 100 terms. Terms were selected by evaluating the *tfIdf* values of terms. Terms that had a *tfIdf* value of between .05 and 1 were chosen.

The largest clusters were evaluated first, and documents in those clusters were removed from further consideration. Once all three term candidate clusters were considered, the process was repeated with two term candidate cluster sets. Mock also proposed an extension that allowed overlapping clusters. In this model terms were not eliminated until all clusters were created at a given level. Our implementation utilizes overlapping clusters.

III. THE REVERSE TREE MODEL

As in Mock's model an Apriori algorithm with an item set lattice to a depth of three terms was utilized. Term selection was performed by selecting a term threshold, or minimum support, and removing stop words. Candidate clusters were again generated by a brute force method.

Candidate clusters were then evaluated at the three term level. While a minimum cluster size was selected for a candidate cluster set to be considered valid, all documents were considered at a given lattice depth. This allowed cluster overlap or fuzziness among clusters.

IV. DATA SOURCE AND PREPERATION

The standardized collection of 21578 Reuter's news articles, as complied by Dr. David D. Lewis, Ph.D. [4], was utilized in this study. This collection of articles represents a subset of Reuter's articles that were published in 1987. This subset consists of articles from three time periods; Spring, Summer, and Fall. The articles are contained in a set of 21 files.

A goal of this study was to develop a clustering system that would lend itself to rapid clustering of documents or web pages. It was therefore elected to develop a clustering system based solely on the title data. This corpus consisted of few common terms. Only 11 terms after limited stemming occurred in more than 5% of the titles. As the title data consist of a short set of terms, this would simulate the terms that may be listed for a web page meta content under keywords or description.

A sample subset of 2% of the articles was created that demonstrated the following inconsistencies with the data:

1. Most articles were formatted in uppercase, but not all.
2. A large number of numeric values existed (i.e. 10, 1,000,000, 5.3, etc). Furthermore, the formatting of the numeric values was not consistent.
3. The tense of verb was not consistent.
4. Plurals of nouns occurred.
5. Multiple spellings for the same word occurred.
6. Words that referred to similar topics (i.e. Japan and Japanese).
7. Non consistent combination of words (i.e. German-mark, German/mark).

Review of the numeric values contained within the titles demonstrated that in many cases the numeric values were nouns and not adjectives. Salton, in his review on term selection recommended the use of nouns and certain verbs as ideal for use in clustering. Numeric values were therefore translated into string values for given ranges that were consistent with how those values were utilized as nouns. This is summarized in Table I.

TABLE I.
String conversions of numeric values.

Numeric Range	Replacement Term
Values < 10	INT
Values < 100, Value ≥ 10	TENS
Value<1000, Value ≥ 100	HUNDREDS
Value<100,000, Value≥1,000	THOUSANDS
Value ≥100,000	BIG

The stemming of the titles consisted of translating ranges of numeric values into terms (Table I), removing stop words, and limited term consolidation. In doing so, no formal stemmer algorithm was utilized on the terms.

An evaluation of the titles of the entire set of 21,578 articles revealed that they contained 13,744 distinct terms, with a total count of 138,337 terms.

It was elected to utilize a document frequency of 30 to 1,079 documents for significant terms. Only 11 terms occurred in more than 1,079 documents. This is believed to reflect the limited length of any given title in the corpus. The final terms set consisted of 643 terms.

A minimum document frequency of 30 was selected based on a graphical review of term selection. At document frequencies of less than 30, there was an exponential increase in the number of terms selected. Figure 1 demonstrates the number of terms selected versus the document frequencies.

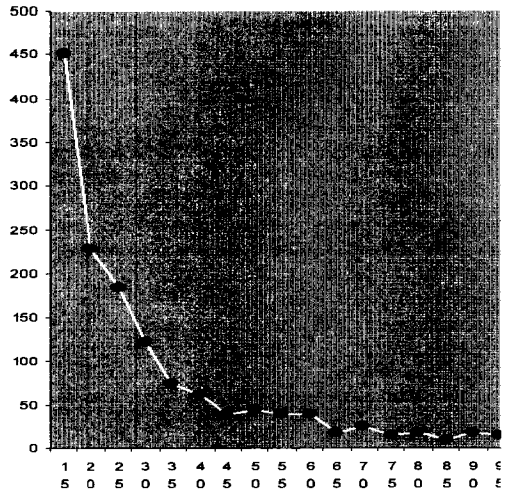


Figure 1. Term Selection v. Document Freq.

V. REVERSE TREE ALGORITHM STRUCTURE

The characteristics of the proposed algorithm are based on a Boolean determination as to whether a document contains a key term. Documents were not weighted as to the number of occurrences of a term within them or the document length. If a document contained a key term, the id number of the document was associated with the term in a Java LinkedList Object. For example a terms set is represented as:

LinkList=$\{d_a, \ldots, d_z\}$

And a cluster is represented as:

$$Cluster = LinkList_1 \bigcap LinkList_2$$

Intersections of the LinkedList object were performed at the three term cluster level. The number of documents present in a candidate set, to be considered valid, was varied in the evaluation as discussed below. A document could be present in more than one candidate set. Allowing documents to be present in more than one candidate set, there by introducing fuzziness into the algorithm.

Documents that existed in a least one valid three term cluster were then removed from further consideration.

The remaining documents were evaluated in a similar manner at the two term level. Every three term cluster had three possible two term roots and these two term clusters were added to the three term leaf. Two term clusters that did not have a three term leaf formed new clusters. The same number of documents, or minimum support, had to be present in the two term candidate sets as in the three term sets for that set to be considered valid.

Documents that existed in a least one valid two term cluster were then removed as before. The remaining documents were then evaluated at the one term level. These were processed as above. Any document not in a cluster after this phase was considered an outlying document.

VI. ALGORITHM IMPLEMENTATION

The implementation of the algorithm required two passes through the document data. The first pass compiled a set of terms present in the set and their frequency. The second pass associated documents with the generated candidate terms. Total time for execution of these two passes was 80 seconds. All processing times for implementation utilizing Sun Java 1.4.2, an AMD 2400Mhz processor and 1GB or ram.

The candidate sets at the three term level were then evaluated by a brute force method. This did require the potential generation of 6,099,006 candidate sets. The number of candidate sets is only a potential, as the intersection of the LinkList was performed in series. This method was felt to be more robust than support based pruning for there was little memory overhead in the execution. Secondary to the above, two term clusters had to be evaluated de novo after creating the three term clusters.

Various minimum cluster sizes were evaluated. The degree of overlap of clusters or their fuzziness was inversely proportional to the minimum number of documents required in a candidate set to be considered valid. The result of varying the minimum support for cluster formation is demonstrated in Table II.

TABLE II
Cluster Formation v. Minimum Support

Min. Cluster Size	3	20
1 Term Clusters	601	518
2 Term Clusters	1867	352
3 Term Clustes	6949	109
Docs. Not Clustered	932	967

VII. CONCLUSSION AND FUTURE EVALUATION

The Reuter's document set of 21578 articles presents a minimum number of documents that a algorithm must be capable of processing in a robust manner as current business needs extend to document sets of 2-3 million [5].

The K-means algorithm is frequently used as a tool in clustering of data. There exist several limitations in utilizing the standard *K-Means* algorithm to evaluated text data.
A primary problem that was appreciated when evaluating the Reuter's data was how to determine the number of clusters that exist in the data.

Two traditional methods of solving the problem are hierarchical agglomerative clustering and bisecting k-means. Both of these algorithm solve the problem, but at a high operational overhead. This limits the usefulness of either algorithm when presented with a large document set.

A secondary problem is the requirement of document clustering algorithm to allow overlap of clusters of a Fuzzy Algorithm. While Fuzzy K-Means variants exist, they require apriori knowledge of the number of clusters to create.

The Reverse Tree algorithm presented addresses both of these issues. No foreknowledge is required of the number of clusters to create and fuzziness is allowed. While the Reverse Tree algorithm is not a vector space model, a comparison can be made.

A random evaluation of the clusters formed did reveal suitable clustering. Neither the cohesion nor the F values based on previous classification of the documents were considered.

While the above results demonstrated that this implementation of a Reverse Decision Tree algorithm provided a suitable method of clustering of the Reuter's news articles, it was not found to be satisfactory for human interface. A report of clusters generated, with one article number per line resulted in a document over 1900 pages in length.

This problem could be addressed with a tree type GUI interface. As this was a preliminary evaluation of the algorithm, such a GUI was not generated.

One solution to the above was to generate a program in which the user could enter terms to form custom searches. Using the same term LinkList Objects that were used to generate the clusters, the user could generate a set of clusters that match their inquiry. Run times for this user guided clustering were less than one second.

A reverse tree rule set generated a satisfactory classification algorithm for a large data source in reasonable time. Supervised learning was required to generate a list of synonyms and stop words for document terms, as was it required in creating a threshold number of occurrences for a term to be significant. The use of an array of term LinkedLists allowed for rapid execution of the clustering and a workable user interface.
On going evaluation of the reverse tree algorithm is currently under way on data sets of larger text documents. Other models of term selection are under review that allows robust run times on the document set as well.

Referemces

[1] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques", University of Minnesota, Technical Report #00-034 (2000). http://www.cs.umn.edu/tech_reports/

[2] Gerard Salton, *Automatic Text Processing*, chapter 5, Massachusetts: Addison-Wesley, 1989.

[3] K. Mock, "A Comparison of Three Document Clustering Algorithms: TreeCluster, Word Intersection GQF, and Word Intersection Hierarchical Agglomerative Clustering". Intel Technical Report 9/23/1998.

[4] Lewis, D., Available: http://www.daviddlewis.com/resources/testcollections/reuters21578/

[5] Person communication—Compulit, Grand Rapids, MI.