Clustering by Agglomeration of Fuzzy Object Granules with Application to Web Personalization

by

Liyu Yi

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado
Date 1/11/00

Signed: Liyu Yi
Liyu Yi

Approved: K Raghavan
Dr. Raghu Krishnapuram
Professor
Thesis Advisor

Golden, Colorado
Date 1/11/00

Graeme Fairweather
Dr. Graeme Fairweather
Professor and Head,
Department of Mathematical and
Computer Sciences

ABSTRACT

In this thesis, we propose a new clustering algorithm called "Clustering by Agglomeration of Fuzzy Object Granules" (CAFOG) for Web mining application. CAFOG differs from its traditional peers by its ability to deal with outliers and with overlapping clusters of arbitrary shape. It is a combination of three clustering methods: Leader Clustering (LC), Fuzzy c-Medoids Clustering (FCMdd), and Fuzzy Agglomerative Clustering (FAC). All three are relational clustering algorithms, thus enabling CAFOG to handle a variety of Web objects which cannot be represented by numerical vectors.

The LC algorithm is a one-pass incremental clustering algorithm. We use it in the first stage in an effort to reduce the size of the input data set for the subsequent stages. It groups objects into a large number of small sub-granules. Its output is a set of leaders (or representative objects), where each leader represents a sub-granule. FCMdd works on the sub-granule representatives generated by LC to carry out the main task of CAFOG. It needs only a few iterations to eliminate most of the noise and outliers, and it generates clean granules (sub-clusters) as its output. Each granule is represented by a medoid. FCMdd requires the specification of the number of granules (sub-clusters), which is chosen to be much higher than the expected number of clusters in the data. In the last stage, FAC uses a modified version of the

well-known Agglomerative Hierarchical Clustering (AHC) [34] algorithm to merge appropriate sub-clusters and automatically determine the number of clusters. The overall complexity of CAFOG is linear in the number of objects.

To illustrate the performance of CAFOG, we apply it to Web personalization. In particular, we use it to organize the Web snippets returned by a search engine proxy in response to a query. Our method consists of scanning snippets returned by the search engine in one pass, and recording the statistical information in a "Lex Tree". We then use the Inverted Document Frequency (IDF) to select significant keywords, and group the keywords into several "concept clusters" using CAFOG. We apply the well-known Apriori [1] algorithm within each concept cluster to identify frequently occurring "itemsets" (or tuples of keywords), and use the itemsets as subtopics associated with the concept cluster. We present the results of the query to the user in two layers. The top layer shows the concept clusters, and the next layer shows the subtopics along with the snippets. Our experiments demonstrate that CAFOG finds more meaningful clusters than AHC in this application.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my thesis committee members Dr. Raghu Krishnapuram, Dr. Xindong Wu and Dr. Tracy Camp for their efforts in guiding me towards my goal. Specifically, I am grateful to Dr. Raghu Krishnapuram for giving me timely help and support, and for the enlightening discussions. I am grateful to Dr. Wu for teaching me the concepts of data mining and the principles of software engineering. I am also grateful to Dr. Camp for the knowledge I gained from her Computer Networking course. Finally, I would like to thank National Science Foundation for their financial support of this project.

Chapter 1

INTRODUCTION

1.1 Motivation and Problem Definition

With the spread of the Internet and the WWW, a new vision of a ubiquitous (anytime, anywhere) information space is emerging. However, even with a modest number of users, mostly in developed economies, there is a staggeringly large amount of information available on the network and the phrase "information overload" has come into the popular vocabulary. Realizing the vision of distributed knowledge access in this scenario and its future evolution will need tools to"personalize" the information space. The state-of-the-art involves users having to navigate their way through the morass of the Infosphere to reach the page containing the information they want. The search engines of today operate mostly at the syntactic level, and are little more than keyword-based indices into documents. As such, they have no way of distinguishing the meaning or the context of a word in the query. Web personalization and mining will have a significant impact on the way documents are searched for and delivered. Personalization has two components, one is tailoring the content delivered to the user from a Web site, and the other is exploring the available Web pages and categorizing them.

Web personalization/mining can be viewed as the extraction of structure from an unlabeled, semi-structured data set containing the characteristics of users/information. Categories, components, or classes are examples of structure. The process of obtaining a compact description of the input data in terms of components, where each component is represented by a parametrized prototype, is often referred to as model identification. A surprising number of model identification problems can be cast as clustering problems, where the prototype of each cluster represents a parametrized component. Specifically, we can expect clustering Web objects such as Web documents and images into reasonable sub-groups to improve the quality and efficiency of their retrieval. Also clustering query results helps the user to separate the items of interest from the spurious or unanticipated so that the query can be further refined or expanded. One important assumption implied in the previous observation is the so-called "Cluster Hypothesis" which says that Web objects that are relevant to a request are more likely to be similar to one another than to irrelevant objects.

In most practical applications, the clusters are not crisp, i.e., they do not have well-defined boundaries. They have considerable overlap. Fuzzy clustering methods provide a better description tool compared to crisp techniques when the cluster boundaries are not crisp. They also have a lesser tendency to get stuck in local minima.

Clustering algorithms need to be robust if they are to be useful in real-world

situations. By robustness, it is meant that the partition generated by clustering should not deteriorate drastically due to noise and outliers. In the case of Web personalization/mining, outliers in the data set can easily occur due to a variety of reasons. Robust and efficient clustering is ill-posed, and is exacerbated by the fact that when the number of components is unknown, a given algorithm may well identify noise points as a component.

The capability of clustering algorithms to deal with relational objects is also important. In real-world problems, not all the objects can be represented by feature vectors. In many cases, the only information available for these objects is of relational nature, e.g., dissimilarities between every pair of objects. Finally, clustering algorithms will be more efficient if they are incremental, i.e., they can process data objects as soon as they arrive rather than waiting for all of them to arrive. The incremental aspect requires that the clustering algorithm be able to make "local" decisions on part of the data.

In summary, to be suitable for Web personalization/mining applications, a clustering algorithm must have the following characteristics:

1. It should be fuzzy rather than crisp.

2. It should be robust and be able to deal with noise and outliers.

3. It should be able to deal with relational data.

4. It should be incremental.

5. It should have low computational complexity.

Unfortunately, most traditional clustering algorithms do not possess one or more of the characteristics listed above.

1.2 Proposed Approach

To overcome the weaknesses of the traditional clustering algorithms, we explore a robust fuzzy clustering algorithm called "Clustering by Agglomeration of Fuzzy Object Granules" (CAFOG). CAFOG has all the desirable characteristics and can improve both the effectiveness and efficiency of the clustering results. The highlights of CAFOG are as follows:

1. *Leader Clustering (LC)*

   Leader Clustering is the first stage of CAFOG. LC is a pre-clustering process that incrementally takes a large input data set and generates sub-granules, where each sub-granule is represented by a leader. The linear complexity and "one-pass" nature of LC makes CAFOG an incremental algorithm that is capable of dealing with large data sets.

2. *Robust Fuzzy C-Medoids (RFCMdd) Clustering*

RFCMdd is the next stage of CAFOG. It is an iterative process that generates a specified number of fuzzy granules (sub-clusters). Most of the noise/outliers will be eliminated in this step. Although it is a global and iterative process, it converges very fast. RFCMdd works as the main body of CAFOG and takes advantage of the fuzzy approach to generate high quality sub-clusters. RFCMdd represents each fuzzy object granule (sub-cluster) by a prototypical object selected from the subcluster. The prototypical object is also known as the medoid.

3. Fuzzy Agglomerative Clustering (FAC)

   FAC is the final stage of CAFOG. It takes the subclusters produced by RFCMdd as its input. FAC is a modified version of the well-known Agglomerative Hierarchical Clustering (AHC) [34] algorithm. FAC has a more effective dissimilarity measure which takes into account the overlap of the subclusters. FAC also automatically determines the number of final clusters. FAC produces complex clusters where each cluster can have multiple representatives. The representatives are the medoids of the fuzzy object granules (subclusters) that comprise the complex clusters.

1.3 Application to Web Personalization

We also design a search engine proxy that uses this new clustering algorithm to cluster the responses to a given query. The search engine proxy is a real Web

application using our new clustering algorithm. It is built as a CGI application that works together with an Apache Web Server on a Linux platform. A typical scenario of the search engine proxy is as follows:

1. The user invokes the search engine proxy by submitting a HTML form to the CGI program, which is an ordinary query the user may send to any regular search engine, such as Yahoo.

2. The search engine proxy forwards the query to one or more regular search engines (we use Yahoo in this implementation), and collects the responses from the regular search engine.

3. The search engine proxy clusters the responses using CAFOG.

4. The search engine proxy returns the clustered responses to the user agent.

The search engine proxy turns out to be a good demonstration of CAFOG. It is more than a simple application of CAFOG. It includes several innovations. First, we introduce an efficient data structure called "Lex Tree" to collect the statistical information about the responses. Second, the search engine proxy uses a new presentation format. Instead of clustering the responses directly, we identify high-level "concepts" in the responses and the subtopics related to each high-level concept. The responses to the query are presented in two layers. The top layer presents the main concepts, and the second layer displays the subtopics and the individual Web-links and snippets

assigned to the subtopics. We use CAFOG to identify the main concepts, and the well-known Apriori [1] algorithm to identify the subtopics.

1.4 Organization of Thesis

The rest of this thesis is organized as follows. Chapter 2 surveys the relevant literature . We give the detailed discussion of CAFOG in Chapter 3. Section 3.2 is about LC. Sections 3.3 and 3.4 present FCMdd and its robust version RFCMdd. We discuss FAC in Section 3.5. We also give some examples involving synthetic and real data in Chapter 3. In Chapter 4, we present a Web personalization application of CAFOG, i.e., a search engine proxy. Finally we give a brief summary of the thesis in Chapter 5.

Chapter 2

LITERATURE SURVEY

2.1 A Brief Review of Clustering Algorithms

2.1.1 Introduction

Clustering aims at classifying the objects in a data set into different groups, such that members of the same cluster are as similar to one another as possible, while members of different clusters are as dissimilar to one another as possible. Because no *a priori* knowledge about the class labels is assumed, clustering is also called unsupervised classification. Several approaches to clustering exist, and differ vastly from each other because they originated in different domains of artificial intelligence and statistics. For example, graph-theoretic and tree-based techniques are popular in the machine learning community, while objective function-driven or prototype-based clustering methods (such as K-Means) [5] have been used in pattern recognition. Clustering algorithms themselves can be classified into different categories based on the type of input they deal with, the type of clusters they consider or the methods they use to construct the result. Here we briefly review these categories.

- Object Clustering **vs.** Relational Clustering

Some clustering algorithms can only deal with object data. Object data means that the objects to be clustered can be represented by n-dimensional feature vectors. However, in many cases, it may not be possible to represent the objects to be clustered by feature vectors. The only available information may be about the relations among the objects, e.g. the dissimilarity matrix which contains the "dissimilarity" between every pair of objects in the data set. Relational data is considered to be more general. Given an object data set and a pre-defined dissimilarity measure, we can always generate the corresponding relational data, but not vise versa. Clustering algorithms that can deal with relational data are called relational clustering algorithms.

- Hierarchical Clustering vs. Partitional Clustering

  Hierarchical clustering methods generate a hierarchy of clusters. There is a single cluster at the root level which is divided into several clusters in the first level. Clusters in the first level are again divided into several subclusters in the next level, and so on. Hierarchical clustering can be agglomerative of divisive. Agglomerative hierarchical clustering algorithms start with each single object as a cluster. In each subsequent step, they merge some of the clusters to generate larger clusters (to be placed in the next higher level) based on certain criteria. This process is repeated until a stopping condition is satisfied. In contrast, partitional clustering methods generate a single partition of the

data, and not a hierarchy. Partitional clustering methods usually optimize an objective function. Many of them are iterative. In each iteration, the clustering algorithm improves the partition based on the objective function. The iterations terminate when a local minimum of the objective function is reached or until a specified maximum number of iterations is reached.

- Prototype Clustering **vs.** Non-prototype Clustering

  Clustering algorithms can also differ by whether they use prototypes to represent clusters or not. There are several different ways to choose prototypes. The simplest prototype is the centroid, which is the mean of all the object vectors in the cluster. The centroid can be computed only if we have object data. For relational data, we can select one of the objects in the cluster as the prototype. Sometimes we may need to use multiple prototypes instead of a single prototype to represent a cluster.

- Fuzzy Clustering **vs.** Crisp Clustering

  Crisp Clustering generates a partition in which each object belongs to only one subset (or cluster). This may not always be the best way to model the real world problem. In contrast, fuzzy clustering allows an object to belong to multiple clusters. Each object is bound to each cluster to a certain degree, $u \in [0, 1]$, also known as "membership". It is well-known that complex data sets containing overlapping clusters are better modeled by fuzzy partitions than by their "crisp"

(non-fuzzy) counterparts. In particular, fuzzy memberships are richer than crisp (binary) memberships in describing the degrees of belonging of objects lying in regions of overlap. Thus, crisp clustering is always a special case of fuzzy clustering. Moreover, fuzzy objective functions are smoother and generally make the optimization process less prone to local or sub-optimal solutions.

2.1.2 K-Means and Fuzzy c-Means Clustering Algorithms

Most prototype-based clustering methods are derivatives of the K-Means [5] algorithm and its fuzzy counterpart, namely the Fuzzy c-Means(FCM) [5] algorithm. In this section, we briefly review these two algorithms.

Let $X = \{\mathbf{x}_j | j = 1, \ldots, n\}$ be a set of feature vectors in a d-dimensional feature space, where $\mathbf{x}_j = [x_{j1}, x_{j2}, \ldots, x_{jd}]$. Let $V = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c)$ represent a c-tuple of prototypes each of which characterizes one of the c clusters. The K-Means algorithm minimizes the objective function given below.

$$J(V, \mathbf{U}; X) = \sum_{i=1}^{c} \sum_{\mathbf{x}_j \in X_i} d^2(\mathbf{x}_j, \mathbf{v}_i), \tag{2.1}$$

In (2.1), $d^2(\mathbf{x}_j, \mathbf{v}_i) = d_{ij}^2$ represents the distance from an object (feature vector) $\mathbf{x}_j$ to prototype $\mathbf{v}_i$. X_i, the i-th cluster, is given by

$$X_i = \{\mathbf{x}_j \in X | d_{ij}^2 = \min_{1 \le k \le c} d_{kj}^2\} \tag{2.2}$$

The optimal prototypes V are derived by setting $\frac{\partial J}{\partial V} = 0$. For instance, if d_{ij}^2 is the squared Euclidean distance, i.e., $d_{ij}^2 = ||\mathbf{x}_j - \mathbf{v}_i||^2$, then the center $\mathbf{v}_i$ is given by

$$\mathbf{v}_i = \frac{\sum_{\mathbf{x}_j \in X_i} \mathbf{x}_j}{n}. \tag{2.3}$$

The K-Means algorithm starts by randomly picking c prototypes. It then updates the partition and prototypes in an alternating fashion using (2.2) and (2.3) until convergence.

K-Means generates a crisp partition, where each point $\mathbf{x}_j$ belongs to only one cluster. In contrast, the Fuzzy c-Means algorithm generates a fuzzy partition. With a fuzzy partition, a data object $\mathbf{x}_j$ belongs to cluster X_i, to a degree $u_{ij} \in [0, 1]$ called fuzzy membership. A fuzzy partition, usually represented by a $c \times n$ matrix $\mathbf{U} = [u_{ij}]$, is called a constrained fuzzy c-partition of X, if the entries of $\mathbf{U}$ satisfy the following constraints [11]:

$$0 \leq u_{ij} \leq 1 \quad \forall i, j, \tag{2.4}$$

$$\sum_{i=1}^{c} u_{ij} = 1 \quad \forall j = 1, \ldots, n, \tag{2.5}$$

and

$$0 \leq \sum_{j=1}^{n} u_{ij} \leq n \quad \forall i = 1, \ldots, c. \tag{2.6}$$

The Fuzzy c-Means(FCM) [5] algorithm minimizes:

$$J_m(V; X) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^m \; d_{ij}^2 \tag{2.7}$$

The optimal prototypes V are derived by setting $\frac{\partial J}{\partial V} = 0$. For instance if d_{ij}^2 is the squared Euclidean distance, i.e., if $d_{ij}^2 = \|\mathbf{x}_j - \mathbf{v}_i\|^2$, then the center $\mathbf{v}_i$ is given by

$$\mathbf{v}_i = \frac{\sum_{j=1}^{n} u_{ij}^m \mathbf{x}_j}{\sum_{j=1}^{n} u_{ij}^m}. \tag{2.8}$$

The optimal FCM memberships for (2.7) can be shown to be [5]

$$u_{ij} = \frac{\left(\frac{1}{d_{ij}^2}\right)^{\frac{1}{m-1}}}{\sum_{k=1}^{c} \left(\frac{1}{d_{ij}^2}\right)^{\frac{1}{m-1}}}. \tag{2.9}$$

Therefore, the optimization process consists of alternating updates of the memberships as given by (2.9), and the cluster centers as given by (2.8). The fuzzy memberships allow each data point to belong to all clusters to a varying degrees. Hence, fuzzy partitions involve less commitment than their hard or crisp counterparts in every step

of the optimization process. This results in a lower sensitivity to initialization. By changing the distance measure and prototypes in the objective function of FCM, it can be generalized to seek clusters of various shapes such as lines, curves and planar quadric surfaces [5, 25].

Both K-Means and FCM use centroids to represent clusters in the clustering process. Therefore, they are prototype-based clustering algorithms that can only deal with object data. There is also a relational relative of FCM called the Relational Fuzzy c-Means (RFCM) algorithm that can deal with relational data. Hathaway and Bezdek [16] derived this algorithm by modifying the FCM objective function.

2.1.3 Agglomerative Hierarchical Clustering (AHC)

The Agglomerative Hierarchical Clustering (AHC) [34] algorithm starts with each isolated object as a subcluster in the bottom-most level. In each step, it looks for the "closest" pair of subclusters and merges them to generate an agglomerated cluster at the next higher level of the hierarchy. We obtain a fewer and fewer number of clusters as the clusters grow larger. The AHC algorithm is summarized below:

The Agglomerative Hierarchical Clustering (AHC) Algorithm

Select a dissimilarity measure $d(X_i, X_j)$ between clusters X_i and X_j;
Select number of final clusters c;
Initialize n = number of input objects; $c' = n$;
Initialize clusters $X_i = \{\mathbf{x}_i\}$,
    where $i = 1, \ldots, n$, and $\mathbf{x}_i$ denotes the i-th input object;

6 **Repeat**
7 $c' = c' - 1$;
8 *Find the closest cluster pair* (X_i, X_j);
9 *Merge* X_i *and* X_j;
10 **Until**{ $c == c'$ }

There are four most commonly used distance measures determining the closest cluster pair in Line 8 of the algorithm,

$$d_{min}(X_i, X_j) = \min\{d(\mathbf{x}_i, \mathbf{x}_j) | \forall \mathbf{x}_i \in X_i \text{ and } \forall \mathbf{x}_j \in X_j\}, \tag{2.10}$$

$$d_{max}(X_i, X_j) = \max\{d(\mathbf{x}_i, \mathbf{x}_j) | \forall \mathbf{x}_i \in X_i \text{ and } \forall \mathbf{x}_j \in X_j\}, \tag{2.11}$$

$$d_{avg}(X_i, X_j) = \frac{\sum_{\mathbf{x}_i \in X_i} \sum_{\mathbf{x}_j \in X_j} d(\mathbf{x}_i, \mathbf{x}_j)}{n_i n_j}, \text{and} \tag{2.12}$$

$$d_{mean}(X_i, X_j) = d\left(\frac{\sum_{\mathbf{x}_i \in X_i} \mathbf{x}_i}{n_i}, \frac{\sum_{\mathbf{x}_j \in X_j} \mathbf{x}_j}{n_j}\right). \tag{2.13}$$

In (2.10)-(2.13) $d(X_i, X_j)$ denotes the distance between cluster X_i and X_j, $n_i = |X_i|$ and $n_j = |X_j|$ represent the cardinalities of X_i and X_j. The first three

dissimilarity measures only require relational information. The resulting relational versions of AHC are called Single Linkage, Complete Linkage, and Average Linkage, respectively. These algorithms do not use any prototypes to represent the clusters during the clustering process. Therefore, they are non-prototype-based relational clustering algorithms. The forth distance measure defines the distance between two clusters as the distance between their centroids. Since we need to compute the centroids, the objects must be represented by feature vectors. Hence, the distance measure in (2.13) makes AHC a prototype-based hierarchical clustering algorithm that can only deal with object data.

One advantage of Single Linkage is that it can be implemented using an efficient Minimum Spanning Tree (MST) algorithm [39]. This is done as follows. We build the Minimum Spanning Tree that links all the objects together. An edge e_{ij} (between two objects $\mathbf{x}_i$ and $\mathbf{x}_j$) in the MST has a weight w_{ij} associated with it. The weight is equal to the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$, and it also represents the dissimilarity $d(X_i, X_j)$ between two subclusters where $\mathbf{x}_i \in X_i$ and $\mathbf{x}_j \in X_j$. Given the number of final clusters c, we can determine the dissimilarity threshold T_D to produce c clusters. We then cut all the edges in the MST with weights larger than T_D. This results in the specified number of final clusters. Potential problems of this algorithm are well-known [14].

2.1.4 Other Related Clustering Algorithms

Several other clustering algorithms have been proposed in the more recent literature. Here we provide a brief review of some of them.

Zhang, Ramakrishnan and Livny [41] proposed a hierarchical clustering algorithm called Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH). It is especially suitable for very large databases. Its I/O cost is linear in the size of the data set. BIRCH maintains a triple (number of data points, linear sum of the data points and squared sum of the data points) for each cluster called Clustering Feature (*CF*) that summarizes the cluster's statistical information. BIRCH builds a *CF* tree in a finite-sized memory by a single scan of the input data set. Every node in the *CF* tree that associates with a *CF* triple represents a subcluster at a particular level of the hierarchy. Finally AHC is applied to the *CF* vectors globally on a specific level of the *CF* tree. The major contribution of BIRCH is its formulation of the clustering problem in a way that is appropriate for very large data sets, by making the time and memory constraints explicit. However, BIRCH can only deal with object data.

Guha, Rastogi and Shim [14] proposed another hierarchical clustering algorithm for large databases called Clustering using Representatives (CURE). CURE employs a hierarchical clustering algorithm that adopts a middle ground between using the centroid and using all points to represent a cluster. It claims to alleviate the shortcomings of the all-points-based as well as the centroid-based approaches. In CURE,

a specified number of well-scattered points in a cluster are first chosen. The scattered points capture the shape and extent of the cluster. The positions of the chosen scattered points are then "shrunk" towards the centroid of the cluster by a fraction α. These shrunken points are used as representatives of the cluster. The clusters with the closest pair of representative points are the clusters that are merged in each step of CURE's modification of AHC. To handle large databases, CURE also employs a combination of random sampling and partitioning. CURE can only deal with object data.

Another well-known relational clustering algorithm is PAM (Partitioning Around Medoids) due to Kaufman and Rousseeuw [23]. This algorithm is based on finding k representative objects (also known as *medoids* [22]) from the data set in such a way that the sum of the within-cluster dissimilarities is minimized. A modified version of PAM called CLARA (Clustering LARge Applications) to handle large data sets was also proposed by Kaufman and Rousseeuw [23]. Ng and Han [29] propose a variation of CLARA called CLARANS. This algorithm tries to make the search for the k representative objects (medoids) more efficient by considering candidate sets of k medoids in the neighborhood of the current set of k medoids. However, CLARANS is not designed for relational data. Finally, it is also interesting to note that Fu [12] suggested a technique very similar to the k medoid technique in the context of clustering string patterns generated by grammars in syntactic pattern recognition.

COBWEB [17] is another incremental clustering algorithm that is popular in the data mining community. Some of the more recent algorithms for relational clustering include [4, 13, 31, 35]. However, these algorithms are not designed for very large data sets and they are not robust.

In summary, we see that there are no clustering algorithms that are suitable for Web mining. They all have one or more of the following drawbacks: (i) high computational complexity, (ii) inability to deal with relational data, (iii) low robustness with respect to outliers, (iv) inability to deal with data incrementally, and (v) inability to deal with overlapping clusters.

2.2 Web Personalization and Mining

2.2.1 Introduction

Data Mining (DM) and knowledge discovery (KD) aim at automatically discovering or extracting knowledge from large amounts of unstructured or semi-structured data. Web mining deals with extracting knowledge about Web objects and users from data such as Web documents, snippets and access logs. Possible applications of robust fuzzy clustering in the field of Web mining include (i) categorization of the Web documents and snippets for efficient search, (ii) categorization of users for customized presentation of Web sites and Web documents, and (iii) grouping of search results for effective and personalized display. We now briefly review a few important papers

related to these three applications.

2.2.2 Document Clustering

Chen *et al.* [7] presented an integrated approach to fuzzy information retrieval which combines techniques in fuzzy set theory with methodologies in textual retrieval in order to achieve optimal retrieval performance. To capture the relationships among index terms, fuzzy logic rules are used. In their paper, several fuzzy clustering (such as FCM and fuzzy hierarchical clustering) methods are adapted to the task of clustering documents with respect to the index terms (or keywords). The clusters generated provide a basis for building the fuzzy logic rules. The clusters are also used to form hyper-links between documents. The advantage of this method is the emphasis on semantic information which is supposed to lead to superior performance.

In another paper by Anick and Vaithyanathan [2], the authors describe how to construct a context-based retrieval system automatically by using document clustering and phrasal analysis. *Context* is defined as a cluster of logically related articles (i.e., the document's "extension") and a small set of salient concepts represented by words and phrases and organized by the cluster's key terms (i.e., the document's "intension"). The authors argue that the focused relevance feedback provided by contexts, at a level of abstraction higher than individual documents and lower than the database as a whole, provides a better way for users to refine vague information needs, and helps to blur the distinction between searching and browsing. They also

argue that linguistic units, such as noun phrases, make better units for describing cluster content than a list of single words. This paper also provides a *Paraphrase* interface, running over a database of business-related news articles, to illustrate the advantages of the context-based retrieval paradigm.

2.2.3 Mining User Access Patterns from Logs

Nasraoui *et al.* [27] explore the use of Web access logs to mine typical user profiles. The authors define the notion of a *user session* as being a temporally compact sequence of Web accesses by a user. They also define a new dissimilarity measure between two Web sessions that captures the organization of the Web site being mined. A relational fuzzy clustering algorithm which can automatically determine the number of clusters is used to cluster the sessions. This method was successfully used to analyze server access logs. Each cluster generated by the method corresponds to a typical session profile. For example, from a university Web site, they extracted the profile of "general outside visitors", the profile of "prospective students", the profile of "students attending a specific class," and so on.

2.2.4 Identifying Frequently Occurring Sets of Keywords in Documents

The *Apriori algorithm* [1] is commonly used for mining "association rules" in data mining. The association rules are generated from sets of items that frequently

occur together. These sets are also known as frequent itemsets. The Apriori algorithm makes use of the concept that every subset of a frequent itemset must also be a frequent itemset. The algorithm proceeds iteratively, first identifying frequent itemsets that contain just one item (i.e., 1-itemsets). In each subsequent iteration, the frequent itemsets identified in the previous iteration are augmented with another item to generate larger candidate itemsets. By only considering itemsets obtained by enlarging previously found frequent itemsets, the number of candidate frequent itemsets is significantly reduced.

In [36], Srikant and Agrawal proposed two efficient algorithms called Cumulate and EstMerge for mining generalized association rules in large databases. These are improved versions of Apriori. Given a large database of transactions where each transaction consists of a set of items, and a taxonomy (*is-a* hierarchy) on the items, associations between items can be found at any level of the taxonomy. For example, given a taxonomy that says that jackets *is-a* outerwear *is-a* clothes, the rule "People who buy outerwear tend to buy shoes" may hold even if its more general or more specific versions "People who buy clothes tend to buy shoes" and "People who buy jackets tend to buy shoes" do not hold.

2.2.5 Personalization of Search Engine Response

With the proliferation of Web servers and pages, Web search engines have become increasingly ineffective tools for locating appropriate content in response to a given

query. Typical queries elicit hundreds (if not thousands) of URLs in response, forcing the user to wade through them in order to find the URL(s) she needs. In large part, this can be attributed to the following:

- *The words involved in the search have multiple meanings.* For example, a user searching for "cricket" may be interested in either the sport or the insect.
- *The user's desired search cannot easily be captured by keywords alone.* For example, a user is looking for Web pages about work on mobile computing done by a particular group of people.

Notice that these problems are independent of how good the algorithms are, that associate keywords with the contents of a page.

One possible solution to this problem is to realize that the responses from search engines to a particular query can be broadly grouped into meaningful categories. If the user is shown these groups, possibly with some descriptive titles, she can select one (or more) which fits her perceived interests. There has been prior work along these lines, such as that by Croft [8], and more recent work by Cutting *et al.* [9]. However, this prior work is in the context of general text collections.

Some earlier Web personalization systems also include the Firefly system [19] which attempted to provide CDs that best match a user's professed interests. Systems such as W^3IQ [21] and PHOAKS [37] have sought to use cooperative information

retrieval techniques for personalization. The Webwatcher project [3] at CMU highlights hyper-links in a page based on the declared interests and the path traversal of a user as well as the path traversals of previous users with similar interests.

Information filtering by using "Hotlists" and "Coldlists" is a method of Web personalization proposed by Pazzani, Nguyen and Mantick [30]. Their research describes a software agent facilitated by three different learning approaches: Bayesian Classifiers, Nearest Neighbor algorithms and Decision Trees. They use these approaches to find information on the Web and decide what new pages might interest a given user. The learning algorithms require a set of positive examples and a set of negative examples represented by feature vectors. They generate "Hotlists" and "Coldlists" of keywords by computing their probability and similarity or by making partition decisions. After the learning phase, each unseen page (that is also represented by a feature vector) will be classified to be either "of interest" or "of uninterest" by checking against the "Hotlists" and "Coldlists". This information can be used to inform the user when a new interesting page becomes available or to order the user's exploration of unseen existing links so that the more promising ones can be investigated first.

Chang, Hsu *et al.* provide another Web Mining application based on document clustering and concept-based relevance feedback [6]. The basic idea is to organize documents retrieved by the original query into conceptual groups so that the user

can get a quick overview of what the query has retrieved in the minimal amount of time. Under this design philosophy, document clustering becomes the first step in concept-based feedback. The target of the research that is introduced in this paper is to implement a concept-based feedback interface in a personalized Web information search assistant by integrating existing search engines with techniques of query mechanisms of keyword extraction for both cluster digesting and query expansion.

Recent work of Etzioni *et al.* [40] also deals with the notion of clustering Web search engine results. They propose an algorithm called Suffix Tree Clustering (STC) for forming groups of "snippets" (or summaries) of Web pages. Essentially, this algorithm uses standard techniques from the literature which allow the construction of "suffix trees" in time that is linear in the number of snippets (or document snippets) assuming that the number of words in each snippet is bounded by a constant. Each node in this tree captures a phrase (or a certain "suffix") of the snippet string, and has associated with it those snippets which contain it. These nodes are viewed as base clusters since they group together all documents that have one phrase in common. Each cluster is assigned a score based on the number of URLs in the cluster as well as the size of the phrase that they have in common. In order to account for the fact that Web pages in the same group may have more than a phrase in common, they then create a graph which has as its vertices the clusters identified by the suffix tree. They define a binary similarity measure between the clusters which is set to

one if at least half of the documents in each cluster are common to both. Vertices representing similar clusters are connected by an edge. They then run a connected component finding algorithm, and each connected component of the graph is identified as a group of documents that are similar. One problem with the STC approach is that the complexity of constructing the graph is exponential in the number of keywords. Thus, it becomes inefficient when the number of keywords is large.

Chapter 3

CAFOG: A NEW ROBUST FUZZY CLUSTERING ALGORITHM FOR LARGE DATA SETS

3.1 Introduction

In this chapter, we give a detailed discussion of the CAFOG algorithm. Section 3.2 is about the Leader Clustering (LC) algorithm, which acts as the pre-clustering process to generate sub-granules of data objects. In Sections 3.3 and 3.4, we discuss the Fuzzy c-Medoids Clustering algorithm (FCMdd) and the Robust Fuzzy c-Medoids (RFCMdd) Clustering algorithm, either of which can function as the main body of CAFOG. They group the sub-granules of the data objects generated by LC into larger fuzzy granules (subclusters). RFCMdd has the additional property that it can identify and eliminate outliers. After (R)FCMdd, we present the Fuzzy Agglomerative Clustering (FAC) algorithm in Section 3.5. FAC is used in the last stage to determine the final clusters. In the last section of this chapter we present several examples to illustrate LC, (R)FCMdd and FAC.

3.2 Leader Clustering (LC) Algorithm

LC is used as a simple pre-clustering process to generate sub-granules of data objects. The aim is to decrease the load in the subsequent stages by reducing the input size, while at the same time minimizing the risk of degrading the quality of the final result. The advantages of LC are that it is both simple and local, which means it can be carried out quickly and incrementally. In other words, we do not need to wait for all the data objects to arrive before we start processing. Also, the process finishes in one pass. However, a pre-set dissimilarity threshold needs to be selected for this algorithm. Whenever an object falls beyond the dissimilarity threshold, the object is considered to be a new leader that will start a new sub-granule. The Leader Clustering algorithm is summarized below:

The Leader Clustering (LC) Algorithm

Select a dissimilarity measure $d(\mathbf{x}_i, \mathbf{x}_j)$ *between objects* $\mathbf{x}_i$ *and* $\mathbf{x}_j$,
  and a threshold T_L;
Read the fist input object $\mathbf{x}_1$;
Initialize set L *of Leaders:* $L = \{\mathbf{l}_1\}$, *where* $\mathbf{l}_1 = \mathbf{x}_1$;
Initialize $i = 2$; $n_L = |L|$; $w_1 = 1$;
Repeat
  Obtain the i*-th input object* $\mathbf{x}_i$;
  $q = \underset{1 \le k \le n_L}{\text{argmin}}\, d(\mathbf{x}_i, \mathbf{l}_k)$;
  If $d(\mathbf{x}_i, \mathbf{l}_q) > T_L$
    $\mathbf{l}_{n_L+1} = \mathbf{x}_i$;
    $L = L \cup \{\mathbf{l}_{n_L+1}\}$;
    $n_L = n_L + 1$;
    $w_{n_L+1} = 1$;
  Else

Assign $\mathbf{x}_i$ *to Leader* $\mathbf{l}_q$;
$w_q = w_q + 1$;
Endif
$i = i + 1$;
Until{ *No more input*}

At the end of the algorithm, we simply use the n_L leaders as the representatives of the data set. Each leader represents a sub-granule, and the weight w_i associated with the sub-granule denotes the number of objects assigned to the leader. We do not use the non-leader objects in subsequent processing. If the weight of a sub-granule is very low, then we can consider the sub-granule as a noise (or outlier) sub-granule and delete it.

The computational complexity of the algorithm is: $\mathcal{O}(n_L n_I)$, where n_I is the number of input objects and n_L is the number of leaders. The price paid for the LC process is that if we reduce the size of the input data set too much, then the resulting granularity is very coarse, which may lump dissimilar objects into the same sub-granule. Also, the clustering result is sensitive to the order in which the data arrive. However, these problems can be partially resolved by the last pass refinement, which will be described in Section 3.6. To be safe, normally a relatively small threshold value is desirable.

3.3 Fuzzy c-Medoids Algorithm

Let $X = \{\mathbf{x}_i | i = 1, \ldots, n\}$ be a set of n objects or leaders. Each object $\mathbf{x}_i$ has a weight value w_i associates with it, and the object may or may not be represented by a feature vector. Let $r(\mathbf{x}_i, \mathbf{x}_j)$ denote the dissimilarity between object $\mathbf{x}_i$ and object $\mathbf{x}_j$. Let $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c\}, \mathbf{v}_i \in X$ represent a subset of X with cardinality c, i.e., V is a c-subset of X, and the elements of V are representatives or medoids of the data set X. Let X^c represent the set of all c-subsets V of X. The Fuzzy c-Medoids Algorithm (FCMdd) minimizes:

$$J_m(V; X) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^m \; w_j r(\mathbf{x}_j, \mathbf{v}_i). \tag{3.1}$$

The minimization is performed over all V in X^c. In (3.1), u_{ij} represents the fuzzy [5], or possibilistic [15, 26] membership of $\mathbf{x}_j$ in cluster i. The membership u_{ij} can be defined heuristically in many different ways. For example, we can use the Fuzzy c-Means (FCM) [5] membership model (see Section 2.1.2) given by:

$$u_{ij} = \frac{\left(\frac{1}{r(\mathbf{x}_j, \mathbf{v}_i)}\right)^{1/(m-1)}}{\sum_{k=1}^{c} \left(\frac{1}{r(\mathbf{x}_j, \mathbf{v}_k)}\right)^{1/(m-1)}}, \tag{3.2}$$

where $m \in [1, \infty)$ is the "fuzzifier". For a given $\mathbf{x}_j$, the above equation can be used only when $r(\mathbf{x}_j, \mathbf{v}_i) \neq 0$, for all i. If $r(\mathbf{x}_j, \mathbf{v}_i) = 0$ for some i, say $i = k$, we use:

$$u_{ij} = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Another possibility (instead of 3.2) is:

$$u_{ij}^m = \frac{\exp\{-Xr(\mathbf{x}_j, \mathbf{v}_i)\}}{\sum_{k=1}^{c} \exp\{-Xr(\mathbf{x}_j, \mathbf{v}_k)\}}. \tag{3.4}$$

Since u_{ij} is a function of the dissimilarities $r(\mathbf{x}_j, \mathbf{v}_k)$, it can be eliminated from (3.1). This is the reason J_m in (3.1) is shown as a function of V alone. When (3.1) is minimized, the V corresponding to the solution generates a fuzzy partition via an equation such as (3.2). However, (3.1) cannot be minimized via the alternating optimization technique, because the necessary conditions cannot be derived by differentiating it with respect to the medoids. (Note that the solution space is discrete.) Thus, strictly speaking, an exhaustive search over X^c needs to be used. However, following Fu's [12] heuristic algorithm for a crisp version of (3.1), we describe a fuzzy algorithm (FCMdd) that minimizes (3.1).

The Fuzzy c-Medoids Algorithm (FCMdd)

Fix the number of clusters c; Set iter = 0;
Initialize the medoids: $V = \{\mathbf{v}_1, \mathbf{v}_1, \ldots, \mathbf{v}_c\}$ *chosen from* X^c*;*
Repeat

for $i = 1$ **to** c **do** /*Compute memberships:*/
  for $j = 1$ **to** n **do**
    Compute u_{ij} *by using (3.2) or (3.4).*
  endfor
endfor
Store the current medoids: $V^{old} = V$;
Compute the new medoids:
for $i = 1$ *to* c **do**
  $q = \underset{1 \le k \le n}{\text{argmin}} \sum_{j=1}^{n} u_{ij}^{m} \, w_j r(\mathbf{x}_j, \mathbf{x}_k)$
  $\mathbf{v}_i = \mathbf{x}_q$;
endfor
$iter = iter + 1$;
Until $(V^{old} = V$ *or* $iter = MAX\_ITER)$.

It can be seen from the last for loop of the FCMdd that the complexity of the algorithm is $\mathcal{O}(n^2)$, where n is the number of input objects or leaders.

We have experimented with three different ways of initializing the medoids. The first way is to pick all the medoid candidates randomly. We call this method Initialization I. The second way is to pick the first candidate as the object that is most central to the data set, and then pick the rest of them one by one in such a way that each one is most dissimilar to all the medoids that have already been picked. This makes the initial medoids evenly distributed. We refer to this procedure as Initialization II.

Initialization II for FCMdd

Fix the number of medoids $c > 1$;
Compute the first medoid:

$q = \underset{1\le j\le n}{\text{argmin}} \sum_{i=1}^{n} r(\mathbf{x}_j, \mathbf{x}_i)$;
$\mathbf{v}_1 = \mathbf{x}_q$;
Set $V = \{\mathbf{v}_1\}$, $iter = 1$;
Repeat
$iter = iter + 1$;
$q = \underset{1\le i\le n; \mathbf{x}_i \notin V}{\text{argmax}} \ \underset{1\le k\le |V|}{\text{min}} \ r(\mathbf{v}_k, \mathbf{x}_i)$;
$\mathbf{v}_{iter} = \mathbf{x}_q$;
$V = V \cup \{\mathbf{v}_{iter}\}$;
Until $(iter = c)$.

For a given data set, the initialization produced by Initialization II is always fixed. Sometimes a bit of randomness might be desirable. In the third initialization strategy, we add randomness by picking the first medoid candidate randomly. The rest of the medoids are selected the same way as in Initialization II. We call this method Initialization III. The computational complexity of Initialization I, II and III are $\mathcal{O}(c)$, $\mathcal{O}(nc^2)$ and $\mathcal{O}(nc^2)$ respectively.

For the sake of completeness, we also provide the crisp version of FCMdd, which we call the Hard c-Medoids (HCMdd) algorithm. In this algorithm, cluster i is denoted by X_i.

The Hard c-Medoids Algorithm (HCMdd)

Fix the number of clusters c; Set iter = 0;
Initialize the medoids: $V = \{\mathbf{v}_1, \mathbf{v}_1, \ldots, \mathbf{v}_c\}$;
Repeat
for $j = 1$ **to** n **do** /*Assign objects to clusters:*/
$q = \underset{1\le k\le c}{\text{argmin}} \, r(\mathbf{x}_j, \mathbf{v}_k)$;

Assign $\mathbf{x}_j$ *to* X_q*;*
endfor
Store the current medoids: $V^{old} = V$*;*
Compute the new medoids:
for $i = 1$ **to** c **do**
$q = \underset{\mathbf{x}_k \in X_i}{\text{argmin}} \sum_{\mathbf{x}_j \in X_i} w_j r(\mathbf{x}_j, \mathbf{x}_k)$
$\mathbf{v}_i = \mathbf{x}_q$*;*
endfor
$iter = iter + 1$*;*
Until $(V^{old} = V$ *or* $iter = MAX\_ITER)$.

Both FCMdd and HCMdd fall in the category of Alternating Cluster Estimation [33] paradigm, and are not *guaranteed* to find the global minimum. It is advisable to try many different initializations to increase the reliability of the results. In practice, the fuzzifier m in FCMdd should be close to 1. A value between 1 and 1.5 is recommended.

3.4 Robust Fuzzy c-Medoids Algorithm

It is well-known that algorithms minimizing a Least-Squares type objective function are not robust [10, 32]. The clustering result can be drastically degraded by the outlier/noise objects. To overcome this problem, we present a variation of FCMdd that is based on the Least Trimmed Squares idea [24].

For the robust version, we use the membership function in (3.2). Substituting

the expression in (3.2) for u_{ij} into (3.1), we obtain:

$$J_m(V;X) = \sum_{j=1}^{n} w_j \left(\sum_{i=1}^{c} (r(\mathbf{x}_j, \mathbf{v}_i))^{1/(1-m)} \right)^{1-m} = \sum_{j=1}^{n} w_j h_j, \tag{3.5}$$

where

$$h_j = \left(\sum_{i=1}^{c} (r(\mathbf{x}_j, \mathbf{v}_i))^{1/(1-m)} \right)^{1-m}. \tag{3.6}$$

When $m = 2$, h_j is $1/c$ times the harmonic mean of the dissimilarities $\{r(\mathbf{x}_j, \mathbf{v}_i)) : i = 1, \ldots, c\}$. The objective function for the Fuzzy c Trimmed Medoids (FCTMdd) algorithm is obtained by modifying (3.5) as follows:

$$J_m^T(V;X) = \sum_{k=1}^{s} w_{k:n} h_{k:n}. \tag{3.7}$$

In (3.7) $h_{k:n}$ represents the k-th item when $h_j, j = 1, \ldots, n$, are arranged in ascending order, $w_{k:n}$ is the corresponding weight, and $s < n$. The value of s is chosen depending on how many objects we would like to disregard in the clustering process. This allows the clustering algorithm to ignore outlier objects while minimizing the objective function. For example, when $s = n/2$, 50% of the objects are not considered in the clustering process, and the objective function is minimized when we pick c medoids in such a way that the sum of the harmonic-mean dissimilarities of 50% of the objects is as small as possible. The objective function in (3.7) cannot be

minimized easily. However, we can design the following heuristic algorithm.

The Fuzzy c Trimmed Medoids Algorithm (FCTMdd)

Fix the number of clusters c, and the fuzzifier m;
Pick the initial set of medoids: $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c\}$ *from* X^c;
$iter = 0$;
Repeat
  Compute the harmonic dissimilarities h_j *for* $j = 1, \ldots, n$ *using (3.6);*
  Sort h_j, $j = 1, \ldots, n$ *to create* $h_{j:n}$;
  Keep the objects corresponding to the first s $h_{j:n}$;
  Compute memberships for s *objects:*
  for $j = 1$ **to** s **do**
    for $i = 1$ **to** c **do**
      Compute $u_{ij:n}$ *by using (3.2);*
    endfor
  endfor
  Store the current medoids: $V^{old} = V$;
  Compute the new medoids:
  for $i = 1$ **to** c **do**
    $q = \underset{1 \le k \le s}{\text{argmin}} \sum_{j=1}^{s} u_{ij:n}^m \, w_{j:n} r(\mathbf{x}_{k:n}, \mathbf{x}_{j:n})$
    $\mathbf{v}_i = \mathbf{x}_q$;
  endfor
  $iter = iter + 1$;
Until $(V^{old} = V$ *or* $iter = MAX\_ITER)$.

The above algorithm is said to use "in-process trimming", because it trims the outliers while it updates the medoids. We also explored an alternative way for trimming outliers. We run the above algorithm with $s = n$. After the process converges, we run FCMdd again, this time with $s < n$. We refer to this alternative version as "post-process trimming". Post-process trimming is computationally faster,

but will give good results only if the outliers are relatively small in number and they are not located very far from the good points. On the other hand, in-process trimming is more sensitive to the choice of the initial V. Thus, many initializations may be required to achieve a truly reliable result.

Again, we caution the reader that the above algorithms can converge to a local minimum. It is good to try many random initializations to increase the reliability of the results. Interestingly, the worst-case complexity of this algorithm still remains $\mathcal{O}(n^2)$. This is a good result, considering that robust algorithms are very expensive. It is quite trivial to design a robust version based on the Least Median of Squares idea as well [28]. In this case we simply replace the summation in (3.7) by the median.

3.5 Fuzzy Agglomerative Clustering

As explained in Section 2.1.3, the basic idea of the AHC algorithm is to merge a pair of subclusters in every step. The pair with the shortest distance between them is selected. However, in our case, the input to the AHC is not a set of regular objects. It is a set of medoids (produced by (R)FCMdd) where each medoid represents a subcluster. The distance between two subclusters X_i and X_k cannot be meaningfully measured by computing the distance D_{ik} between their medoids $\mathbf{v}_i$ and $\mathbf{v}_k$. We need to somehow take in account the affinity between the granules (subclusters) as measured by the trength and extent of the common boundary between them. Since

the membership u_{ij} of an object $\mathbf{x}_j$ in cluster X_i depends on the object's position with respect to the medoid of X_i relative to its position with respect to all other medoids, it indirectly encodes the affinity of $\mathbf{x}_j$ with respect to X_i. Based on this fact, we explored three different ways to define the dissimilarity between two subclusters.

1) **Dissimilarity Modified by overlap**:

Let $U_i = (u_{i1}, \ldots, u_{ij}, \ldots, u_{in})$ represent the fuzzy cluster X_i associated with medoid $\mathbf{v}_i$, where u_{ij} is the degree to which object $\mathbf{x}_j$ belongs to X_i. The overlap between two fuzzy granules (subclusters) U_i and U_k can be measured by the Jaccard index [20]:

$$\frac{|U_i \cap U_k|}{|U_i \cup U_k|}, \tag{3.8}$$

where

$$\begin{aligned} U_i \cap U_k &= (\min(u_{i1}, u_{k1}), \ldots, \min(u_{in}, u_{kn})), \\ U_i \cup U_k &= (\max(u_{i1}, u_{k1}), \ldots, \max(u_{in}, u_{kn})), \\ & \quad and\ |U_j| = \sum_{l=1}^{n} u_{jl}. \end{aligned} \tag{3.9}$$

We can modify the set dissimilarity d_{ik} between X_i and X_k by taking into account

the overlap as follows:

$$D_{ik}^{o} = D_{ik} \left(1 - \frac{|U_i \cap U_k|}{|U_i \cup U_k|}\right). \tag{3.10}$$

2) **Dissimilarity Modified by Bridge Strength:**

$$\text{Let } B_{ik} = \Big\{ \mathbf{x}_j |\ u_{ij} = \max_{1 \le p \le n} u_{pj} \text{ and } u_{kj} = \max_{1 \le p \le n, p \ne i} u_{pj} \text{ or } u_{kj} \max_{1 \le p \le n} u_{pj} \text{ and } u_{kj} = \max_{1 \le p \le n, p \ne k} u_{pj} \Big\}.$$

B_{ik} denotes the set of points for which the top two highest memberships are either in X_i or in X_k, i.e., it is the set of points that lie on the boundary region between X_i and X_k. We define:

$$D_{ik}^{b} = D_{ik} \left(1 - \frac{\sum_{\mathbf{x}_j \in B_{ik}} \min(u_{ij}, u_{kj})}{\sum_{\mathbf{x}_j \in B_{ik}} \max(u_{ij}, u_{kj})}\right) \tag{3.11}$$

3) **Dissimilarity Modified by Gap Strength:**

This measure is defined by

$$D_{ik}^{g} = D_{ik}(1 + G_{max}), \tag{3.12}$$

where G_{max} is the *gap factor*. It measures the gap along the bridge between granules X_i and X_k, which is computed by the following algorithm:

Find the Maximum Gap

Identify the set of bridge objects B_{ik} *between clusters* X_i *and* X_k *using (3.5) which also includes medoid* $\mathbf{v}_i$ *and medoid* $\mathbf{v}_k$.
Let $n_{ik} = |B_{ik}|$ = *cardinality of* B_{ik}.
Sort the bridge objects by $r(\mathbf{x}_j, \mathbf{v}_i) - r(\mathbf{x}_j, \mathbf{v}_k) = r'_j$, *where* $r()$ *is the dissimilarity measure used in (R)FCMdd;*
Obtain ordered list $\{r'_{1:n_{ik}}, \ldots, r'_{n_{ik}:n_{ik}}\}$, *and corresponding list* $\{\mathbf{x}_{1:n_{ik}}, \ldots, \mathbf{x}_{n_{ik}:n_{ik}}\}$;

$$q = \underset{1 \le p \le n_{ik}-1}{argmax} \; (r'_{p+1:n_{ik}} - r'_{p:n_{ik}});$$

$$G_{max} = \frac{(r'_{q+1:n_{ik}} - r'_{q:n_{ik}})}{2d_{ik}}$$

If a pre-set number of clusters c is provided, then the next step of the FAC is very similar to AHC with d_{min} (see Equation 2.10). We build the Minimum Spanning Tree (MST) [39], using one of the three dissimilarity measures D_{ik} presented above. Then we cut some of the edges of the tree to obtain the specified number of clusters (see Section 2.1.3). However, in many applications, we will not be able to specify the number of clusters beforehand. An alternative way to determine the number of clusters is desirable. We do this as follows. After we build the Minimum Spanning Tree (MST) [39], we sort the edges of the MST by D_{ik}. Then we find the largest "jump" between two consecutive D_{ik} to determine the dissimilarity threshold T_D. By cutting the tree at dissimilarity threshold T_D, we automatically determine the number of final clusters. This procedure is shown below.

Algorithm to Find Dissimilarity Threshold

Let n = *number of edges in the MST.*
Sort the edges by D_{ik} *(use one of* D^o_{ik}, D^b_{ik} *and* D^g_{ik}*);*
Obtain list $\{D_{1:n}, \ldots, D_{n:n}\}$*;*
$q = \underset{1<i\leq n}{\text{argmax}}(D_{i:n} - D_{i-1:n})$*;*
$T_D = D_q$

The computational complexity of FAC depends on which dissimilarity measure is used. Let n be the number of input objects to FCMdd. The complexity to compute the overlap between two subclusters is $\mathcal{O}(n)$. Computing the bridge strength is similar to computing the overlap. However, this requires identifying the bridge objects between all pairs of granules (subclusters). Since this is done in FCMdd or RFCMdd at the time of assigning each object to the medoids, no additional computations are required. When we use the dissimilarity measure modified by gap strength, we need to sort all the objects along the bridge. This adds $\mathcal{O}(n \log n)$ computations. Therefore, considering the complexity of building MST, the total complexity of FAC using the dissimilarity measure modified by overlap or by bridge strength is $\mathcal{O}(n_M^2 n)$, where n_M is the number of medoids. The complexity of FAC when we use the dissimilarity measure modified by gap strength is $\mathcal{O}(n_M^2 n \log n)$.

Each cluster generated by FAC contains multiple medoids $\mathbf{v}_i$ which are the representatives of the original granules (subclusters).

3.6 Final Pass Refinement

As we mentioned in Section 3.2, there might be a mis-clustering problem in the LC stage, especially when the specified threshold T_L results in a coarse granularity. An extra global pass can resolve this problem. Therefore, we introduce an optional Final Pass Refinement to improve the quality of the final clusters. The algorithm is summarized below:

Final Pass Refinement

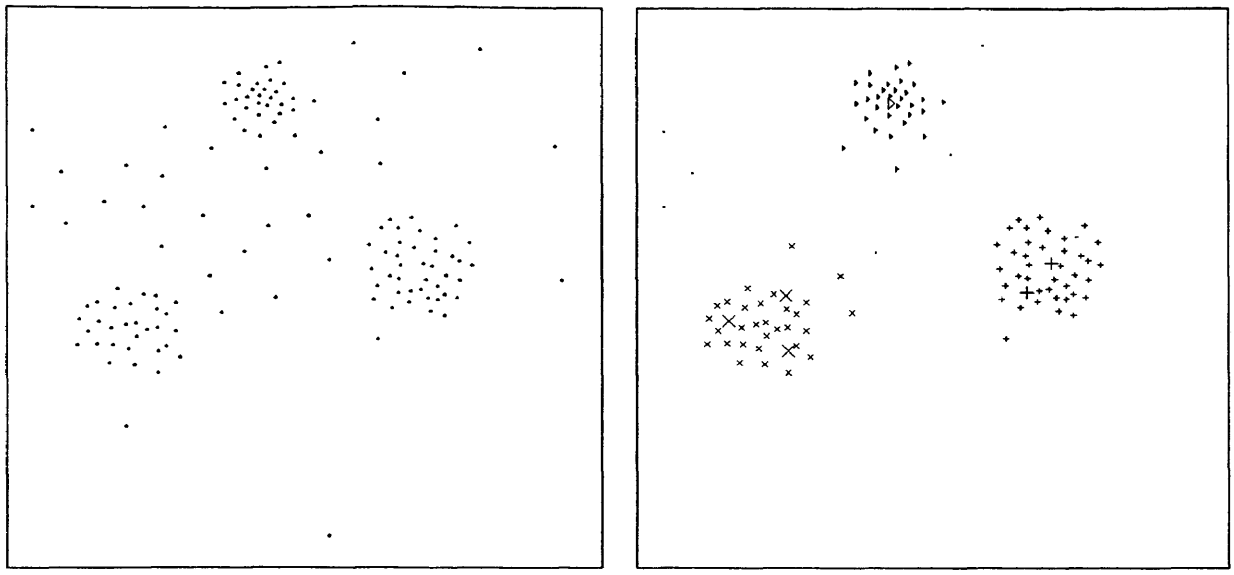
Select a threshold T_M*;*
Let c *be the number of clusters generated by FAC;*
Let V_i *be the set of medoids in the* i*-th final cluster* X_i *generated by FAC.*
Initialize the set of medoids in the i*-th final cluster* $V_i = \{\mathbf{v}_{i1}, \ldots, \mathbf{v}_{ic_i}\}$,
  where $1 \leq i \leq c$*, and* c_i *is the number of representatives of* X_i*;*
Initialize $i = 1$*;*
Repeat
  Obtain the i*-th object* $\mathbf{x}_i$*;*
  $d_{min_k} = \min_{1 \leq j \leq c_k} r(\mathbf{x}_i, \mathbf{v}_{kj}), k = 1, \ldots, c;$
  $q = \operatorname{argmin}_{1 \leq k \leq c} d_{min_k}$;
  If $d_{min_q} \leq T_M$
    $X_q = X_q \cup \{\mathbf{x}_i\}$;
  Else
    Label $\mathbf{x}_i$ *as outlier;*
  Endif
  $i = i + 1$;
Until *No more objects*

3.7 Illustrative Examples

In this section, we use two-dimensional data sets to demonstrate the procedures LC, (R)FCMdd and FAC. The first example is a synthetically generated 2-D data set with 150 objects. It includes 45 outlier objects. In other words, 30% of the data set consists of outliers. Fig. 3.1(a) shows the input to CAFOG and Fig. 3.1(b) shows the final output. In Fig. 3.1(b), different symbols (triangle, cross and plus) are used to denote points in different clusters as determined by CAFOG. Outliers are indicated by simple dots. The dissimilarity measure used in FAC is D_{ik}^{o} (i.e., Euclidean distance modified by overlap) as in (3.10). Post-process trimming was chosen for eliminating the outliers. We can see that CAFOG produces a good result. It determines the intuitively correct number of clusters and also identifies all the outliers. We did not use the Leader Clustering process in this case, because the input size is not very large.

Fig. 3.2(a) shows the input data set for the second experiment, which has 1000 2-D objects in a 2-D region of size 400×400. Fig. 3.2(b) shows the result after LC is applied with $T_L = 10$. It can be seen that LC gives us 364 almost evenly distributed leaders and they represent the data set well.

We then applied the Robust Fuzzy c-Medoids Algorithm on the leaders generated from the LC step. As mentioned in Section 3.3, different initialization strategies can be chosen for RFCMdd. Figs. 3.3 (a)-(c) show the different initializations. The



(a) (b)

FIG. 3.1. 2-D data set with 150 objects including 45 outliers. (a) Input raw data. (b) Output clusters.

bold dots represent the initial medoid candidates selected by the three initialization methods. We can see that Initialization II generates evenly distributed medoids, and Initialization I is generally very uneven, and Initialization III is a compromise. If repeatability of results is important, we should use Initialization II. On the other hand, Many algorithms tend to converge to local minima, and hence repeated use of Initialization III may be reasonable.

The result of Robust FCMdd is shown in Fig. 3.4. The parameters used in Robust FCMdd were as follows: outlier percentage = 15%; number of clusters = 15; fuzzifier m = 1.25; trimming strategy = in-process trimming; dissimilarity measure

= Euclidean distance.

In Fig. 3.4, different symbols are used to represent points belonging to different clusters. (Note that the orientation of the symbol counts.) The larger symbols stand for the medoids, and the dots stand for outliers identified by our algorithm. Fig. 3.5 shows the final result after FAC is applied to the result of RFCMdd. Euclidean distance modified by overlap, i.e., D_{ik}^{o} in (3.10), is used in this stage.

For the sake of completeness, we also present results generated by using various other parameter settings in Figs. 3.6, 3.7 and 3.8. We can see from the figures that in-process trimming and post-process trimming obtain similar resluts. We also see from the figures that with the increase in the specified outlier percentage, we identify more outliers and hence the clusters generated by RFCMdd are of good quality. However, we caution the reader that if the the outlier percentage is too high, then we may have some "good" objects identified as outliers. Since the the RFCMdd is only guaranteed to converge to a local minimum, the best way to obtain a high quality result in the case of heavy noise is to run RFCMdd with multiple initializations.

To compare the performance of FAC and AHC, we manually generated a 2-D data set with clusters of complex shapes. First we used RFCMdd to generate 10 subclusters from this data set, then we applied both FAC and AHC to produce the final clusters. Fig. 3.9(a) shows the mis-clustering of AHC when the traditional Euclidean dissimilarity measure, i.e., $D_{ik} = \|\mathbf{v}_i - \mathbf{v}_k\|^2$ is used. The traditional

dissimilarity measure only relies on the distance between the medoids of the clusters and ignores the "gap" or "overlap" between them. Fig. 3.9(b) shows that FAC with dissimilarity measure D_{ik}^{g} in (3.12) generates a more intuitive result. AHC puts many points from the elongated cluster on the right into the left cluster.

3.8 Examples with Real Data

In this section, we compare HCMdd and FCMdd with Relational Fuzzy c-Means (RFCM) Clustering Algorithm on two Web document data sets. RFCM is a standard relational clustering algorithm derived from FCM (see Section 2.1.2). We also show an example of snippet clustering with FCMdd and FCTMdd.

The first data set is a collection of 1042 abstracts obtained from the Cambridge Scientific Abstract [18] Web site. The abstracts correspond to 10 topics (distance education, filament, health care, intermetallic, laminate, nuclear, aeronautics, plastic, trade, furnace, and recycling). There are about 100 abstracts per topic, but since the abstracts were not carefully chosen, some are outliers. In addition, we deliberately added 20 outliers. The second data set is a collection of 59 HTML documents compiled by 6 students at the Colorado School of Mines. Each student was asked to collect about 10 Web pages related to a particular topic (fish, astronomy, ray tracing, chatroom conversation, neurobiology and sports).

The procedure we used to generate the feature vectors was as follows. We first used a "stop-word elimination and stemming" algorithm obtained from Louisiana State University [7] to filter out insignificant words and remove certain types of word-endings, e.g. "ing" and "ed". We then selected 500 keywords out of the collection by using the Inverted Document Frequency (IDF) method. The IDF method takes into account both the frequency of a given keyword and the information about its

distribution in the whole document collection. The IDF value for a given keyword is computed as follows:

$$f_{idf} = \frac{f_w}{f_{w\_max}} \log\left(\frac{P}{P_w}\right). \tag{3.13}$$

In (3.13) f_w is the frequency of occurrence of the keyword in the document collection, f_{w_max} is the maximum frequency of occurrence of any keyword in the document collection, P_w is the number of documents that include this keyword and P is the number of documents in the whole collection. The IDF value of a keyword measures its "importance". We sort the keywords by their IDF values and select the 500 keywords with the largest IDF values. These keywords are denoted by $K_1, K_2, \ldots, K_{500}$.

We then represent the i-th document by a feature vector $\mathbf{k}_i = [k_{i1}, \ldots, k_{ip}]$, where k_{ik} is the normalized frequency f_d of the the k-th keyword K_k in the i-th document. The normalized frequency $f_d = n_w/n_t$, where n_w is the frequency of the keyword in the document and n_t is the total number of keywords in this document. Thus, we generated a 500-dimensional feature vector for each document. To reduce the dimensionality, we applied principle component analysis (PCA) [38] and selected the eigenvectors corresponding to the top 10 eigenvalues as the new features. Each 500-dimensional vector $\mathbf{k}_i$ representing a document was projected onto the 10 eigenvectors to form a 10-dimensional feature vector $\mathbf{k}'_i$. We perform the clustering on

the dissimilarity matrix generated from these 10-dimensional vectors. In the case of RFCM, the dissimilarity matrix was generated using the square of the Euclidean distance between the vectors $\mathbf{k}_i'$. In the case of FCMdd, several dissimilarity measures were tried.

Table 3.8 shows the results obtained for the first two data sets. The column with the heading FCMdd-cos corresponds to the cases when FCMdd was run with the cosine dissimilarity measure (i.e., 1−cosine of angle between vectors), and the columns with the headings FCMdd-E, FCMdd-E^2, and FCMdd-L_1 correspond to the case when FCMdd was run with the Euclidean distance, squared Euclidean distance, and L_1-norm, respectively. The results in the first row are the average of 20 different runs of the algorithms on the Cambridge Scientific Abstract data set. In each run, 120 abstracts were randomly selected from the collection, and only these 120 abstracts were used to generate the keywords and the 10 eigenvectors. The 500-dimensional feature vectors were constructed for all the remaining abstracts, and these feature vectors were then projected onto the 10 eigenvectors to generate the object data for a particular run of the algorithms. This process was repeated 20 times. The second row of the table corresponds to the Web-page data set.

The results show that FCMdd compares favorably with RFCM in terms of classification rates, and at the same time is an order of magnitude faster. The CPU times (in seconds) were recorded on a Pentium II, 400 MHz processor, and do not include

the time to compute the dissimilarity matrix.

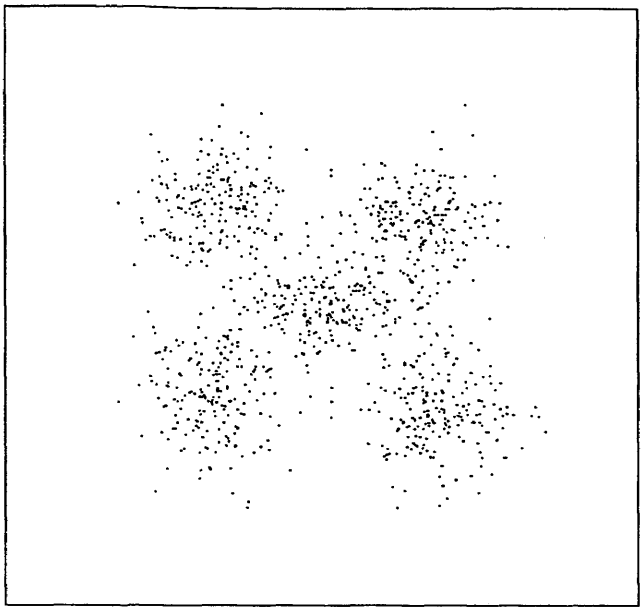
The last example involves a data set consisting of snippets corresponding to 200 Web documents retrieved by a meta search engine [40] in response to the query "salsa". As in the previous example, we used the "stop-word elimination and stemming" algorithm followed by the IDF method to generate $m = 500$ keywords. Each snippet was then represented by a m-dimensional vector $\mathbf{k}_i$. (see previous example.) To compute the dissimilarities $r(\mathbf{k}_i, \mathbf{k}_j)$ between snippets, we used the Jaccard Index dissimilarity [20] measure given by:

$$r_J(\mathbf{s}_1, \mathbf{s}_2) = \frac{\sum_{i=1}^{m} \min(k_{1i}, k_{2i})}{\sum_{i=1}^{m} \max(k_{1i}, k_{2i})}, \tag{3.14}$$

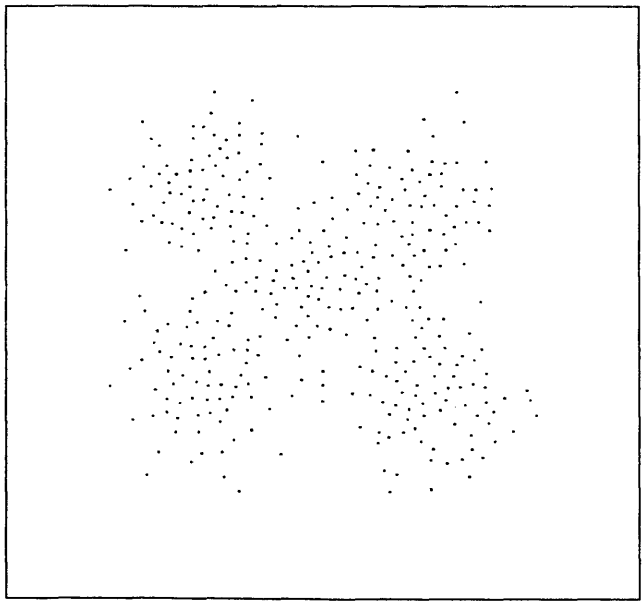
where $\mathbf{k}_1 = [k_{11}, \ldots, k_{1n}]^T$ and $\mathbf{k}_2 = [k_{21}, \ldots, k_{2n}]^T$ are the two m-dimensional vectors representing to two snippets. FCMdd was applied to this data with $c = 7$, and each snippet was crisply assigned to the cluster, in which it had the highest membership (after the algorithm converged). There were three main clusters with 151, 19, and 19 snippets. The remaining clusters had 1 to 6 snippets in them and therefore we did not consider them to be significant. We computed average keyword frequencies for each individual cluster. The top 10 most frequent keywords (along with their average frequencies in parentheses) are shown in Table 3.1. These keywords give us a "profile" of the cluster. As can be seen, the first cluster is mostly about hot sauces, the second one is about salsa music, and the third one is about salsa dancing. Since FCMdd is

not robust, the first cluster also contains many irrelevant (outlier) snippets. Table 3.8 shows randomly sampled snippets from each cluster. It can be seen that the last two snippets in cluster 1 are outliers.

We applied FCTMdd to the same data with 25% in-process trimming, and found that most of the irrelevant snippets were identified correctly as outliers by the algorithm. For example, FCTMdd eliminates the last two snippets shown in Table 3.8 from cluster 1. Cluster 1 now contains only 65 snippets, and the other two clusters are unchanged. Table 3.2 summarizes the results of FCTMdd. If we compare the profile vectors produced by FCTMdd with those of FCMdd, we can see that the profile of cluster 1 is significantly strengthened by FCTMdd. The remaining clusters are virtually unchanged by FCTMdd, since they already have fairly strong profiles.



(a)



(b)

FIG. 3.2. (a) Input data set with 1000 objects. (b) 364 leaders after LC is applied.

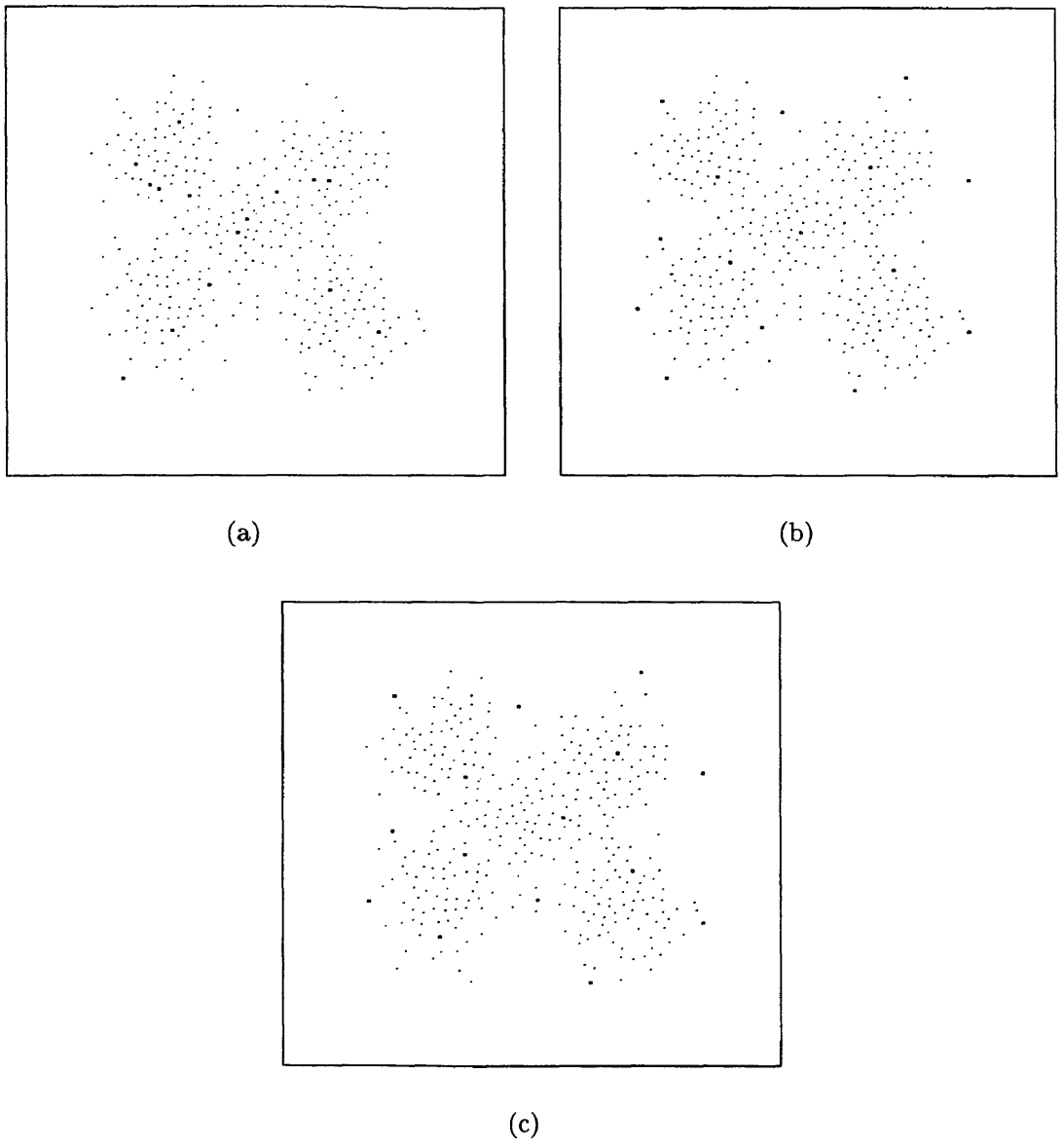


FIG. 3.3. Medoids selected by various initializations. (a) Initialization I. (b)Initialization II. (c) Initialization III.

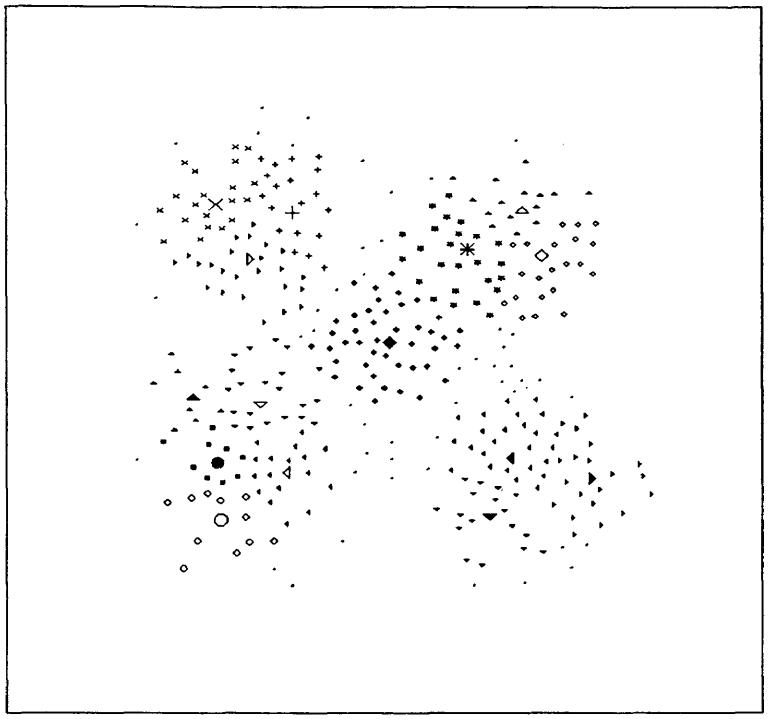


FIG. 3.4. Result of robust FCMdd with an over-specified number of clusters. The outliers are shown as simple dots.

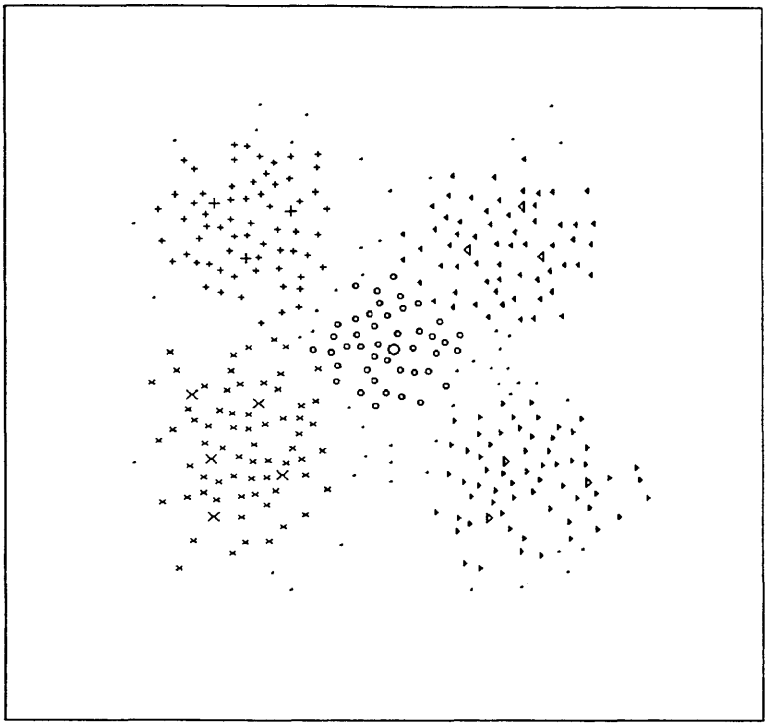
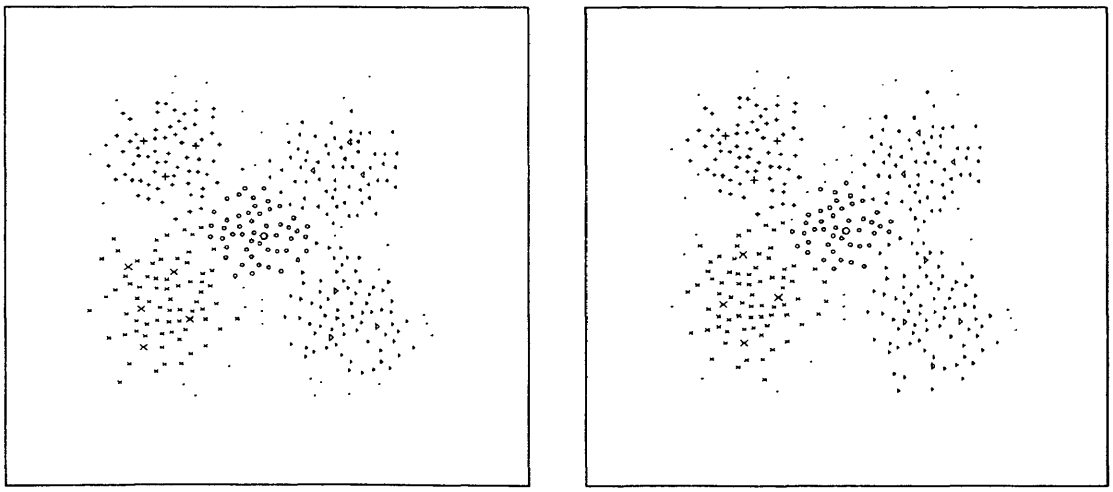
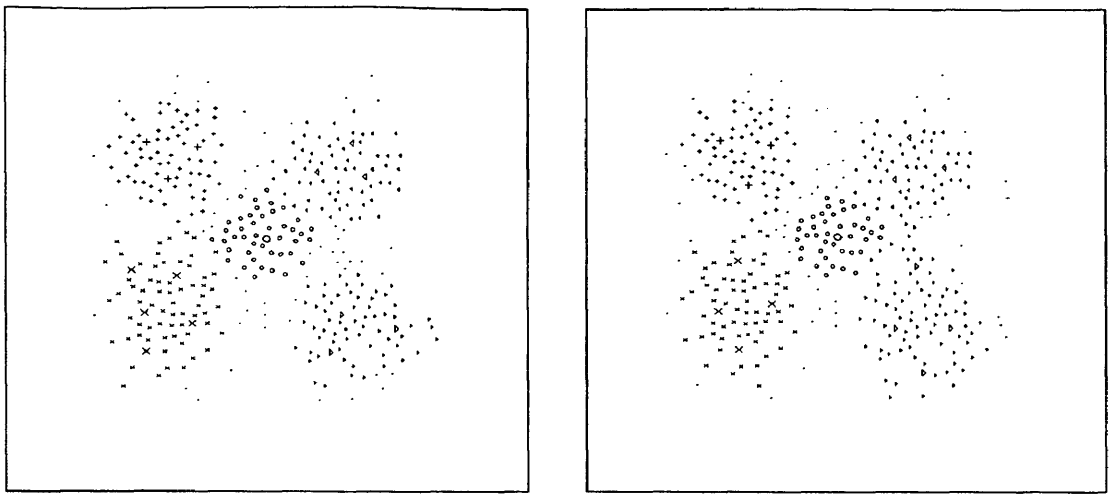


FIG. 3.5. Final result of FAC.



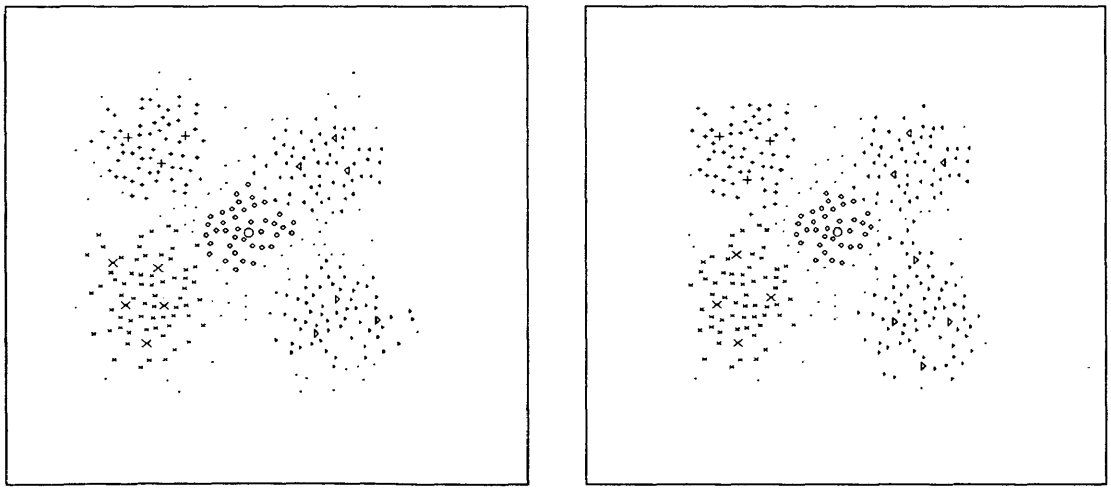
(a) (b)

FIG. 3.6. Final results with *outlier percentage* = 10%: (a) in-process trimming and (b) post-process trimming.



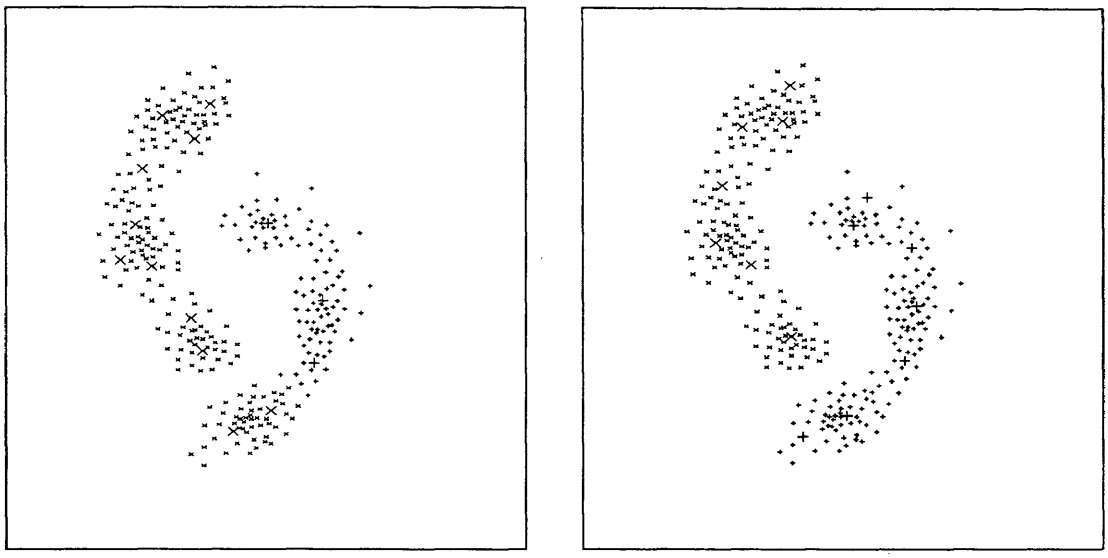
(a) (b)

FIG. 3.7. Final results with *outlier percentage* = 15%: (a) in-process trimming and (b) post-process trimming.



(a) (b)

FIG. 3.8. Final results with *outlier percentage* = 20%: (a) in-process trimming and (b) post-process trimming.



(a) (b)

FIG. 3.9. A comparison of AHC with FAC on a data set with 2 clusters: (a) Result of AHC. (b) Result of FAC using Euclidean dissimilarity measure modified by gap strength.

Table 3.1. Results on Abstract and Web-Page Data Sets.

Data Set	RFCM-E^2		FCMdd-cos		FCMdd-E		FCMdd-E^2		FCMdd-L_1	
	rate (%)	CPU (s)	rate (%)	CPU (s)	rate (%)	CPU (s)	rate (%)	CPU (s)	rate (%)	CPU (s)
Abstr	83.60	126.88	83.10	5.03	74.55	4.20	83.97	5.20	75.52	3.77
Web Pg	83.05	0.34	84.75	0.01	84.75	0.00	84.75	0.01	88.14	0.01

Table 3.2. FCMdd Results on Snippet Data Set.

cluster	kw1	kw2	kw3	kw4	kw5	kw6	kw7	kw8	kw9	kw10
1	hot (.35)	sauc (.23)	thi (.21)	chile (.19)	recip (.15)	food (.13)	gourmet (.13)	garlic (.11)	spice (.10)	mexican (.09)
2	music (1.0)	danc (.63)	dj (.47)	latin (.42)	onli (.26)	todai (.21)	lesson (.16)	listen (.16)	live (.16)	floor (.16)
3	danc (1.4)	lesson (.31)	look (.32)	jeanni (.21)	gonzalez (.16)	jobi (.16)	london (.16)	lui (.16)	oxford (.16)	page (.16)

Table 3.3. FCTMdd Results on Snippet Data Set.

cluster	kw1	kw2	kw3	kw4	kw5	kw6	kw7	kw8	kw9	kw10
1	hot (.82)	sauc (.52)	thi (.48)	chile (.43)	gourmet (.29)	garlic (.25)	food (.23)	recip (.23)	pepper (.19)	mexican (.17)
2	music (1.0)	danc (.63)	dj (.47)	latin (.42)	onli (.26)	todai (.21)	lesson (.16)	listen (.16)	live (.16)	floor (.16)
3	danc (1.4)	lesson (.31)	look (.32)	jeanni (.21)	gonzalez (.16)	jobi (.16)	london (.16)	lui (.16)	oxford (.16)	page (.16)

Table 3.4. Sample Snippets in 3 Clusters Found by FCMdd.

cluster #1
1) Vermont maple syrup, gift baskets, hot sauce, salsa, new England specialty products, gourmet foods, corporate, sweets]. Vermont maple syrup, gift baskets, hot sauce, salsa, new England specialty products, gourmet foods, corporate,
2) HOT SAUCE. #299 Devil's Dozen! $38.00 Click here to see a few of our hot sauces! #201 Batten Island Gourmet, Mild $2.00 closeout - only 1 left #211 Mountain Man Fire Roasted Habanero 3.95$MountainManisourbestsellinghabsauce!$#212$PurpleHaze$3.95
3) Made with searing red savina chiles, hot habanero chiles and thai chiles makes this salsa hotter then you know where. The Paradise Pineapple Salsa was awarded first place in the 1996 Fiery Foods Festival-fruit salsa division.
4) Kaari Tiistai 17.00-18.00 SALSA Alkeet Miguel 18.00-19.00 PARISALSA Alkeet Miguel 19.00-20.00 OPISKELE ESPANJAA Salsaamalla Alkeet (tunti pidet espanjaksi) Miguel 20.00-21.00 RUEDA de CASINO & PARISALSA Jatko Miguel Keskiviikko 17.00-18.00 FLAMENCO Alkeet (sevillanas) Kaari 18.00-19.00 FLAMENCO Jatko (tangos) Kaari 19.00-20.00.
5) By combining a multi-faceted corporate marketing background with the latest in computer animation and presentation techniques, ROUTE 66 PRODUCTIONS, LL-C has formed a division specializing in sophisticated digital media applications, called DIGITAL SALSA.
cluster #2
1) Edinburgh Latin/Hispanic Music and Dance Service: Tel-Aviv Yigal Korolevski has kindly send me a summary of what's going on in Tel-Aviv. Wednesday - 21:00-22:00 (beginners), 22:00-24:00 Brazilian party.
2) A collection of the best music played at Tropicana famous dancing floor. Compositions by Ignacio Pieiro, Ernesto Lecuona, Enrique Jorrn, the father of cha cha ch, Celina Gonzlez, Miguel Matamoros. They received the award "Disco de Oro" for their
3) DJ Jesus R. R. DJ Jesus plays all the hottest Latins tunes: salsa, cumbia, merengue, banda, quebradita, ranchera and romantica. Available to DJ latin techno and salsa hits for any occasion. El Tigre" DJ Manny - LATIN MUSIC PRODUCTIONS. Phone: 283-4213. 3213
cluster #3
1) The instructor, Gustavo Sr. has been teaching authentic Latin dances in the Seattle area for many years, and travels often to his native Panama, to constantly update his knowledge of the true international dance hall scene. The instructor, Gustavo Sr. has
2) What I really love, is watching great salsa dancing - I wish I could dance well but need a lot of lessons and practice (so does my novia who is Guatemalteca). What I really love, is watching great salsa dancing - I wish I could dance well but need a lot of lessons and practice (so does my novia who is Guatemalteca).
3) After a few lessons you'll have stepped into a whole new scene - you'll quickly make new friends, and before long you'll be dancing the nights away at one of Viva Salsa's events or at one of the many salsa clubs in Oxfordshire

Chapter 4

APPLICATION OF CAFOG TO WEB PERSONALIZATION

4.1 Introduction

In this chapter, we describe a practical application of CAFOG to Web personalization. In particular, we consider the problem of presenting the results of a query to the user in a personalized manner. We build a search engine proxy with a few clustering algorithms plugged in. The search engine proxy receives the query from the user agent (browser), and forwards the query to a regular search engine (such as Yahoo). The search results that are returned by the search engine, including the Web links and their summaries (snippets), are collected by the search engine proxy. The search engine proxy picks a specified number of keywords from the collection of summaries based on the IDF method described in Section 3.8. It then applies CAFOG to the Web link summaries to generate clusters of keywords. Each cluster contains a set of keywords, and the set of keywords is assumed to represent a concept. Therefore, we also refer to the clusters as "concept clusters". The search engine proxy applies the Apriori algorithm [1] to the set of keywords in each cluster, and extracts frequently occurring keyword tuples. Each keyword tuple so extracted becomes "a sub-concept" or "a subtopic" of the concept or topic represented by the cluster. The

concept-subtopic hierarchy helps the user to review the results in an easily navigable format. The block diagram of the search engine proxy is shown in Fig. 4.1.

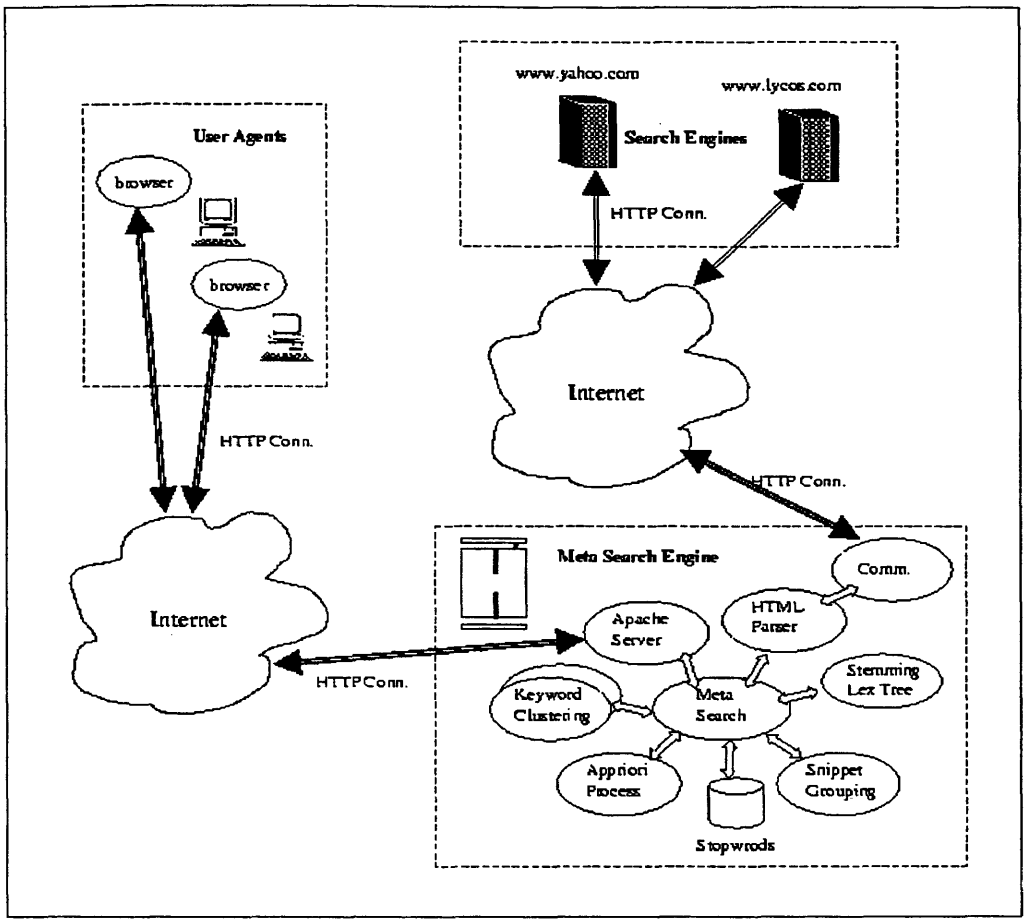


FIG. 4.1. Block Diagram of Search Engine Proxy.

The rest of this chapter is organized as follows. Section 4.2 explains the process of query forwarding. Section 4.3 describes the lexical process that is used to pre-process the data and select keywords. Section 4.4 describes how the keywords are clustered to generate concepts using AHC and CAFOG. Section 4.5 presents the Apriori algorithm we used for locating common keyword tuples. The last section is about generating a

concept hierarchy based on the raw results of the Apriori algorithm.

4.2 Query Forwarding

The search engine proxy is built as a Web application, and uses C++ as the CGI programming language. It receives the query as a "HTML form", and forwards it to the search engine Yahoo. Actually any regular search engine that provides Web-link summaries (snippets) can be used as our data source. We noticed that Yahoo presents a kind of grouping of the query results. However, since these groups are based on statically determined clusters, in most cases this information is not very helpful. To avoid the side effects of Yahoo's grouping, when we forward the query to Yahoo, we use the advanced option which only shows the "WebSites" and forces the relation between the query keywords to "AND". Yahoo returns its results page by page. In order to gather the whole collection of the query results, our search engine proxy submits the HTTP request repeatedly, thus obtaining all the response pages until the maximum number of Web links is reached. In our experiments, this maximum number is set to 1000. At this stage, extra effort is also made to eliminate duplicate Web links in the query responses.

4.3 The Lexical Process

The response from the search engine is treated as the input stream for the lexical process. The Web links are first isolated from the HTML context by parsing out the HTML tags. The URLs and names of the Web links are saved in a dynamically allocated array. The summaries (snippets) are tokenized into words (tokens). A light stemming algorithm [7] is applied on these words to modify or delete the word endings. The stemmed words are inserted into a Lex Tree, which is described below.

The Lex Tree data structure (modified from source obtained from authors of [7]) is the next step of processing the tokenized snippets. The Lex Tree enables us to carry out the procedure in strictly linear time. Lex Tree is a tree structure with several Lex Nodes. Every non-root Lex Node corresponds to a lower-case letter from the Roman alphabet. The path from the root to a Lex Node represents a list of letters. If the list matches any pre-defined stop/noise word then the node is called a Stopword Node. If the non-root node is not a Stopword Node and its path consists of a list that represents a word (token) in one of the snippets, then the node is called a Keyword Node. The remaining non-root nodes are called Branch Nodes. Pre-determined stop words are loaded into the Lex Tree at the initialization stage.

Here we use several figures to explain how the Lex Tree is constructed. The symbols we use in the figures are shown in Fig. 4.2(a). When a Lex Tree is first built, it contains only the Root Node. Then the pre-determined stopwords are inserted.

Fig. 4.2(b) shows the Lex Tree when stopwords "a", "about" and "the" are inserted. We then start to insert the tokenized snippets. Fig. 4.2(c) shows the Lex Tree after snippet "A thesis about the atom" is inserted. In this case, the words "thesis" and "atom" are identified as keywords and chained by a linked list. Note that the words "A", "about" and "the" are stopwords. The result is shown in Fig. 4.2(d).

When a token from the input stream is inserted into the Lex Tree, it proceeds from the Root Node and eventually terminates at a node defining a path. If this path represents a stopword, it means that the token is a stopword. It is then ignored. If the path does not represent a stopword, then the node at which the insertion terminates is either a new Keyword Node or an existing one. In either case, the Keyword Node is updated with new statistical information about the keyword, which is a vector of integers. The i-th element of the vector represents the i-th summary (snippet), and its value is incremented if the keyword just inserted came from Summary i. If this Keyword Node is a newly created node, then it is also added to the link list that chains all the Keyword Nodes together for convenient access. Let K_i denote the keyword represented by a Keyword Node. The vector $\mathbf{k}_i$ associated with the Keyword Node can be thought of as a feature vector representation of K_i.

After scanning all the query responses from the search engine and inserting them into the Lex Tree, the Inverted Document Frequency (IDF) is computed for the keywords identified by the process (see Section 3.8). Based on the IDF value, the

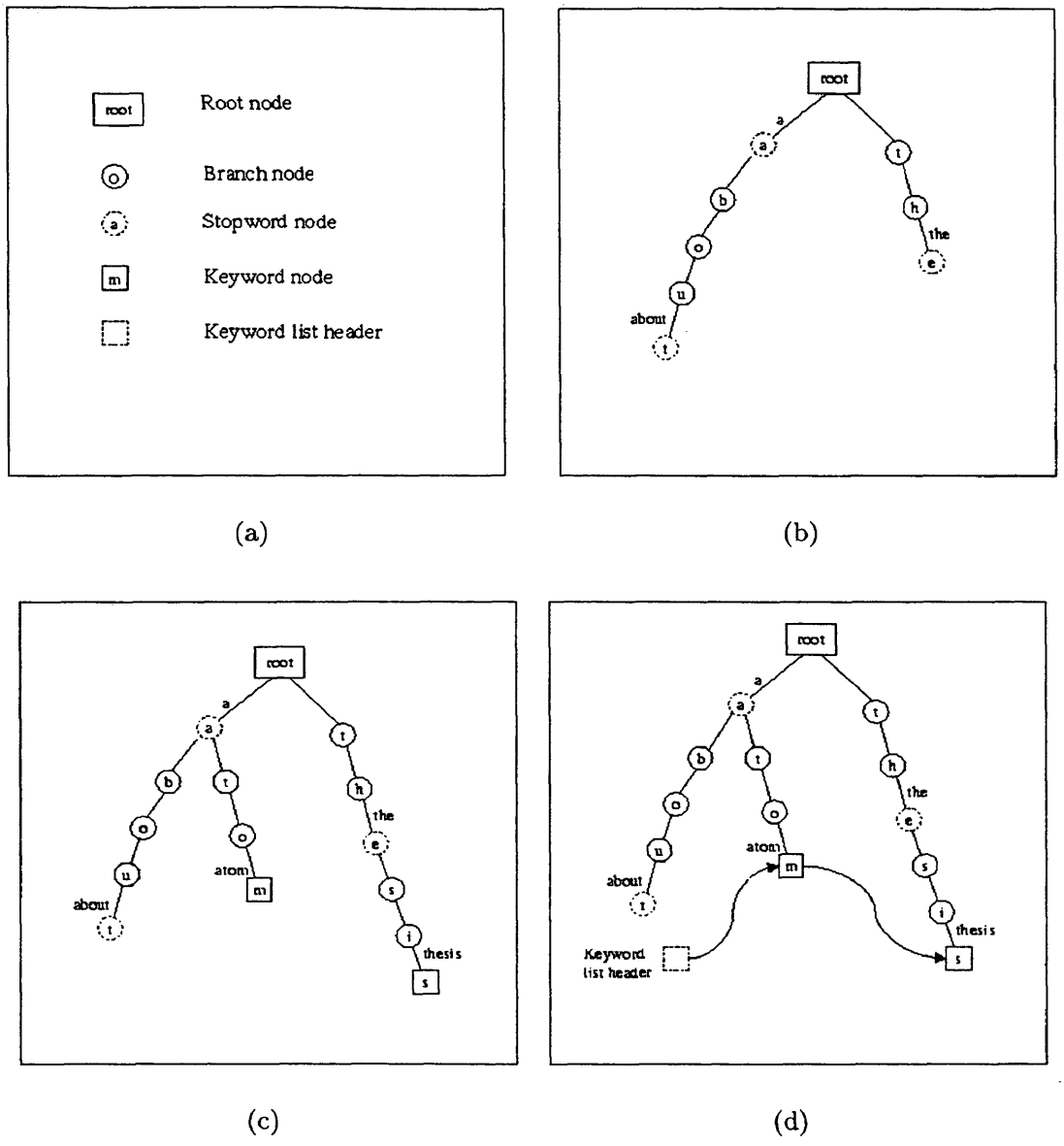


FIG. 4.2. (a) Legend of symbols used in the Lex Tree. (b) Lex Tree after stopwords "a", "about", and "the" are inserted. (c) Lex tree after the tokenized summary "A thesis about the atom." is inserted. (d) Keyword chaining by a linked list.

top m most salient keywords are selected for further processing, where m is a pre-specified number. In our search engine proxy application, we use $m = 200$ keywords. Here we assume that any cluster information pertaining to the current query can be represented by these salient keywords. To conserve memory, at this stage we release the whole Lex Tree except for the Keyword Nodes representing the top m keywords (and the associated feature vectors $\mathbf{k}_i$).

Obviously, the height of the Lex Tree is equal to the length L of the longest stemmed token that appears in the query response, which is a relatively small number. Let n denote the total number of summaries returned by the search engine in response to the query, n_K denote the total number of distinct keywords included in the whole set of summaries, which equals the number of Keyword nodes in the Lex Tree. There is a vector of length n_S associated with each keyword node that records the statistical information about the keyword. Thus, the total memory size required by the Lex Tree can be upper bounded by $\mathcal{O}(n_S n_K)$, assuming the space required by the stopwords is a constant. Let n_T denote the total number of tokens included in all returned summaries. Note that n_T includes the stopwords. The maximum time required to insert each token in the Lex Tree is upperbounded by L, which is a constant. Therefore, the computational complexity to construct the Lex Tree is $\mathcal{O}(n_T)$.

4.4 Generation of Concepts by Clustering Keywords

After the Lex Tree stage, we have a data set of m vectors, where each vector $\mathbf{k}_i = [k_{i1}, \ldots, k_{il}, \ldots, k_{in}]^T$, $i = 1, \ldots, m$, represents a keyword K_i. Element k_{il} of the vector denotes the number of times keyword K_i appears in summary l. Let $\mathbf{k}_i$ and $\mathbf{k}_j$ represent two keyword objects (vectors). We compute the dissimilarity matrix for the keyword objects (vectors) as follows:

$$r(\mathbf{k}_i, \mathbf{k}_j) = \frac{\sum_{l=1}^{n} \min(k'_{il}, k'_{jl})}{n}, \tag{4.1}$$

where n is the total number of summaries and

$$k'_{il} = \begin{cases} 1 & \text{if } k_{il} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

The complexity for generating the distance matrix is $\mathcal{O}(nm)$. To obtain the concepts (topics), clustering is applied on this distance matrix. In our experiment we explored two possibilities for clustering: (1) CAFOG and (2) AHC. In either case, since the input size to the clustering algorithm is an $m \times m$ matrix, which is fairly small, the clustering is quite fast. The cost of generating concept clusters mostly consists of the time needed to compute the distance matrix. Since the number of

keywords is relatively small, we did not use the LC stage in CAFOG.

Besides grouping the keywords to generate topics, the keyword clustering process also narrows down our search for identifying frequently occurring keyword tuples in the next stage. A potential side-effect of clustering before identifying frequently occurring tuples is that if clustering splits related keywords into different clusters, some of the topics may be lost. However, our experiments show that this does not happen in practice. The details are discussed in Section 4.5.

4.5 Locating Frequent Keyword Tuples

The Apriori [1] algorithm is used in our application for identifying frequent keyword tuples, also known as frequent *itemsets*. Each frequent itemset is assumed to be a subtopic.

Let the whole set of items be denoted by $\mathcal{I} = \{i_1, \ldots, i_m\}$, where m is the total number of items in the problem domain. In our case, m is the number of selected keywords. Let the whole set of transactions be denoted by $\mathcal{T} = \{T_1, \ldots, T_n\}$, where n is the total number of transactions. Each transaction T_j is of the form $T_j = \{i_{j1}, \ldots, i_{jn_j}\}$, where $T_j \subseteq \mathcal{I}$, and $n_j = |T_j|$ is the cardinality of T_j. In our application, each transaction represents a summary (snippet). Let $\mathcal{P}(\mathcal{I})$ denote the power set of itemset $\mathcal{I}$. An itemset S_i is an element of $\mathcal{P}(\mathcal{I})$, for $1 \leq i \leq |\mathcal{P}(\mathcal{I})|$. Let $|S_i|$ denote the cardinality of S_i. We denote the set of k-itemsets by $\mathcal{I}_k$, where

$\mathcal{I}_k = \{S_i \subseteq \mathcal{I} \mid |S_i| = k\}$.

Given an itemset S_i, its support is defined as:

$$Support(S_i) = \frac{n_{T_i}}{|\mathcal{T}|}, \tag{4.3}$$

where n_{T_i} is the number of all transactions T_j that satisfy: $S_i \subseteq T_j$. Given a support threshold S_{TH}, frequent itemsets are defined to be those S_i that satisfy $Support(S_i) \geq S_{TH}$. The Apriori algorithm [1] is a method to generate all frequent itemsets of $\mathcal{I}$. The algorithm is summarized below.

Apriori Algorithm

Fix the minimum support of the frequent itemset S_{TH};
Initialize $L_1 = \{S_i | S_i \subset \mathcal{I}_1$ *and* $Support(S_i) \geq S_{TH}\}$;
Initialize $k = 1$;
Repeat
  $L_{k+1} = \Phi$,
  For *all itemsets* $S_i \in L_k$ **do**
    For *all itemsets* $S_j \in L_1$ **do**
      candidate itemset $S_l = S_i \cup S_j$;
      If $(Support(S_l) > S_{TH})$
        $L_{k+1} = L_{k+1} \cup \{S_l\}$,
      End if
    End for
  End for
  $k = k + 1$,
Until $(L_k == \Phi)$;
return $(L_1, \ldots, L_{k-1})$

The algorithm generates a series of sets $L_1, \ldots, L_j, \ldots, L_{k-1}$, where L_j represents the set $\{S_j | \ |S_j| = j$ and $Support(S_j) > S_{TH}\}$. In our case, the frequent

itemsets contained in $L_1, \ldots, L_{k-1}$ correspond to frequently occurring keyword tuples. Every itemset included in $L_1, \ldots, L_{k-1}$ is assumed to be a subtopic. While applying the Apriori algorithm, we determine the minimum support S_{TH} based on the size n of the collection of the responses to the query. We use the following formula:

$$S_{TH} = \begin{cases} 2 & \text{if } n < 60 \\ n/30 & \text{if } 30 \leq n \leq 360 \\ 12 & \text{if } n > 360. \end{cases} \tag{4.4}$$

In theory, the Apriori algorithm is quite expensive when the sizes of $\mathcal{I}$ and $\mathcal{T}$ are not small. In the worst case, its computational complexity is $\mathcal{O}(2^m n)$, where $m = |\mathcal{I}|$ is the number of items (in our case, the number of keywords) and $n = |\mathcal{T}|$ is the size of the set of transactions (in our case, the number of summaries). The worst-case scenario is when almost all $S_i \in \mathcal{P}(\mathcal{I})$ have higher supports than S_{TH}. To alleviate this situation, we can group the keywords into smaller concept clusters and then apply the Apriori algorithm within each of them. This reduces the worst case computational complexity to $\mathcal{O}(2^{m/c} n)$, where c is the number of keyword clusters. However, this conclusion holds only when the clustering process generates clusters of roughly equal size. In general, the more evenly we divide the whole set of keywords, the higher the performance improvement. However, we must also add the cost of clustering, which can be high in relation to the time required to run Apriori for small

data sets.

Since we apply the Apriori algorithm only within each keyword cluster, if some frequent itemsets straddle two or more clusters, we will not find them. However, a "good" clustering result will put similar and related keywords in the same group and dissimilar and unrelated keywords into different groups. This minimizes the possibility of losing frequent itemsets. Our experimental results validate this conjecture.

In this particular application, we noticed that the keyword clustering does not improve the total time requirements, because the number of snippets n is not large enough. Thus, the time saved in Apriori does not compensate for the time required for clustering. However, we also noticed that decreasing the support threshold S_{TH} in Apriori and increasing the number m of selected keywords results in many more frequent itemsets, and a tangible improvement can be seen.

In our actual implementation of Apriori, we made a few more modifications to improve its efficiency further. In line 16 of the algorithm, the candidate itemset of S_i of cardinality $k+1$ is checked against L_k to verify that all of S_i's k-item subsets are elements of L_k.

4.6 Snippet Clustering

After we obtain the subtopics represented by the keyword tuples (frequent itemsets) found by Apriori, the summaries that contain all the keywords in a subtopic are

assigned to that subtopic. By this procedure, a given summary could be assigned to multiple subtopics. However, we place an additional constraint while making the assignment. A summary will not be assigned a subtopic if it has been already assigned to a more specific subtopic. A subtopic represented by itemset S_i is said to be more specific than a subtopic represented by itemset S_j, if $S_j \subset S_i$.

Conceptually, the assignment process can be described as follows. When we apply Apriori, we build a frequent itemset lattice that can be illustrated by Fig. 4.3. The set of all possible itemsets is simply the lattice representation of $\mathcal{I}$. Each node in the lattice denotes a possible itemset. The frequent itemsets found by Apriori are a subset of these nodes, and hence they form a partial lattice. The number of nodes in the partial lattice depends on the support threshold S_{TH} we use in the Apriori algorithm. The top node (Node 0) in Fig. 4.3 corresponds to the empty itemset. In the partial lattice, we can record the links from a k-itemset to all its super itemsets of cardinality $k+1$. An example is shown in Fig. 4.3. When we try to assign a summary to the subtopics, the summary proceeds from the top node and passes through the partial lattice along every possible path. A given node lets the summary through if the summary contains all the keywords in the itemset corresponding to that node. Otherwise, it blocks the summary. The last nodes that allow the snippet's passage are the ones it is assigned to. For example, a summary that includes keywords $\{K_1, K_2, K_3\}$ will be assigned to node 5 and node 3, a summary that includes keywords

$\{K_1, K_2, K_4, K_m\}$ will be assigned to node 7 and node 6.

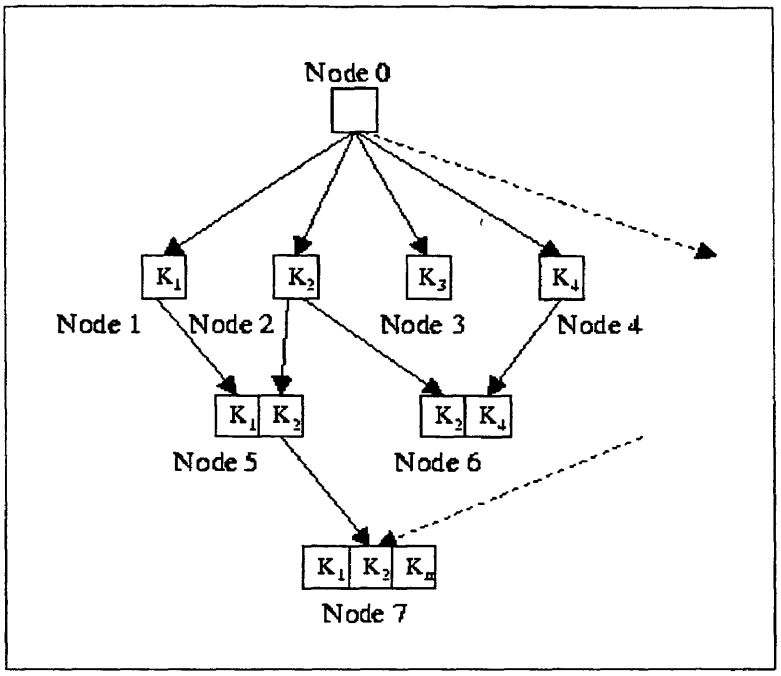


FIG. 4.3. Lattice of subtopics

Since a given vector $\mathbf{k}_i$ records which documents contain the keyword K_i, and these vectors are created in the Lex Tree stage, the assignment step is very efficient. Its computational complexity is $\mathcal{O}(nn_S)$, where n_S is the number of subtopics, and n is the number of summaries.

4.7 Presentation of Results to User

After we generate the concept clusters and assign summaries to the subtopics of the concepts identified by Apriori, we send the results back to the user's browser screen. Our aim is to use the concept-subtopic information to facilitate the user's

browsing. Therefore, we present the results in two layers: (1) the main concept layer where each concept is represented by a set of keywords, and (2) the subtopics layer where relevant Web links and snippets are assigned to each subtopic of a given concept.

We display the main concepts first, because they stand for higher-level and more abstract information. The concept clusters are sorted by their size. We simply assume that the larger clusters are more important and meaningful. Each cluster has a hyper-link to all the subtopics within that cluster. Each subtopic is assigned a group of snippets. The subtopics within each concept cluster are also sorted. The more specific subtopics always go first. A subtopic is said to be more specific if it contains more keywords than others. If two subtopics have an equal number of keywords, then one that has a larger number of snippets assigned to it will be displayed first. The assumption behind this arrangement is that the more specific the subtopic and the larger the number snippets assigned to the subtopic, the more important and meaningful it is. If a snippet falls into a concept cluster but cannot be assigned to any subtopic, it will be displayed under the group labeled "miscellaneous". This happens when a keyword in a concept cluster does not have support larger than S_{TH} to form a subtopic.

To compare the effectiveness of CAFOG, we plugged both CAFOG and AHC into our search engine proxy and examined the concept clusters found. We evaluated

the performance based on the following criteria:

(1) The clustering should assign all the keywords belonging to any given subtopic to the same cluster, so that none of the subtopics are lost when Apriori is applied.

(2) The clustering should divide the keywords as evenly as possible, provided the clusters are meaningful.

In order to evaluate the performance based on Criterion 1, we compare the list subtopics generated when CAFOG or AHC is used with that generated when no clustering is used. In order to evaluate the performance based on Criterion 2, we define a "Good Keyword Cluster" (GKC) to be a "meaningful" keyword cluster with a size larger than 1. The larger the number of GKCs, the better the performance. "Meaningfulness" is judged subjectively, by looking at all the keywords in the cluster to see if they are related.

We collected the results of 20 test queries submitted to the search engine proxy. The queries were deliberately chosen so that they all had multiple interpretations. The query "salsa" was also used in [40]. The LC stage was omitted in CAFOG, FCMdd was applied with 20 clusters, and the dissimilarity measure in (4.1) was used. In the FAC stage, the final number of clusters was automatically determined as described in Section 3.5. The dissimilarity measure used was D_{ik}^{o} (see (3.10)). The results are summarized in Table 4.1. The table shows that neither CAFOG nor AHC loses any subtopic. Apriori generates identical itemsets from the concept clusters produced by

AHC and CAFOG for all queries. We can see that CAFOG generates more GKCs than AHC does. This shows that CAFOG is more effective than AHC in our search engine proxy application.

We also record the typical CPU time required to present the result to the user in all three different configurations. On a $400MHz$ Pentium III PC, the time required when CAFOG is used ranges from 0.1 to 1 second. When AHC is used it ranges from 0.05 to 0.7 second, and when no clustering is used it ranges from 0.02 to 0.3 second.

Table 4.1. Summary of Results on 20 Test Queries.

Queries	No Clustering	AHC		CAFOG	
	No. of subtopics	No. of subtopics	No. of $GKCs$	No. of subtopics	No. of $GKCs$
"salsa"	27	27	2	27	13
"UPS"	57	57	1	57	11
"shuttle"	127	127	2	127	8
"protocol"	31	31	2	31	15
"agency"	110	110	5	110	17
"star"	37	37	2	37	13
"java product"	195	195	1	195	17
"internet publication"	44	44	1	44	15
"chaos"	13	13	3	13	15
"east"	43	43	2	43	10
"chinese food"	35	35	3	35	11
"club"	75	75	1	75	18
"ball"	56	56	2	56	6
"free stuff"	27	27	5	27	15
"artificial intelligence"	67	67	1	67	5
"material science"	121	121	1	121	13
"language research"	58	58	1	58	10
"game"	22	22	2	22	13
"weather report"	27	27	6	27	20
"see"	43	43	3	43	17

Figs. 4.4 and 4.5 are examples of the concept clusters generated by AHC and

CAFOG for the query "UPS". Fig. 4.6 shows the snippet groups assigned to example subtopics generated by either algorithm. There are a total of 57 subtopics found for the query "UPS". Interestingly, we found a snippet group labeled "ups ship", which is about "United Parcel Service", a group labeled "ups power supply", which is about "Uninterruptible Power Supplies" and a group labeled "business start-ups", which is about start-up companies.

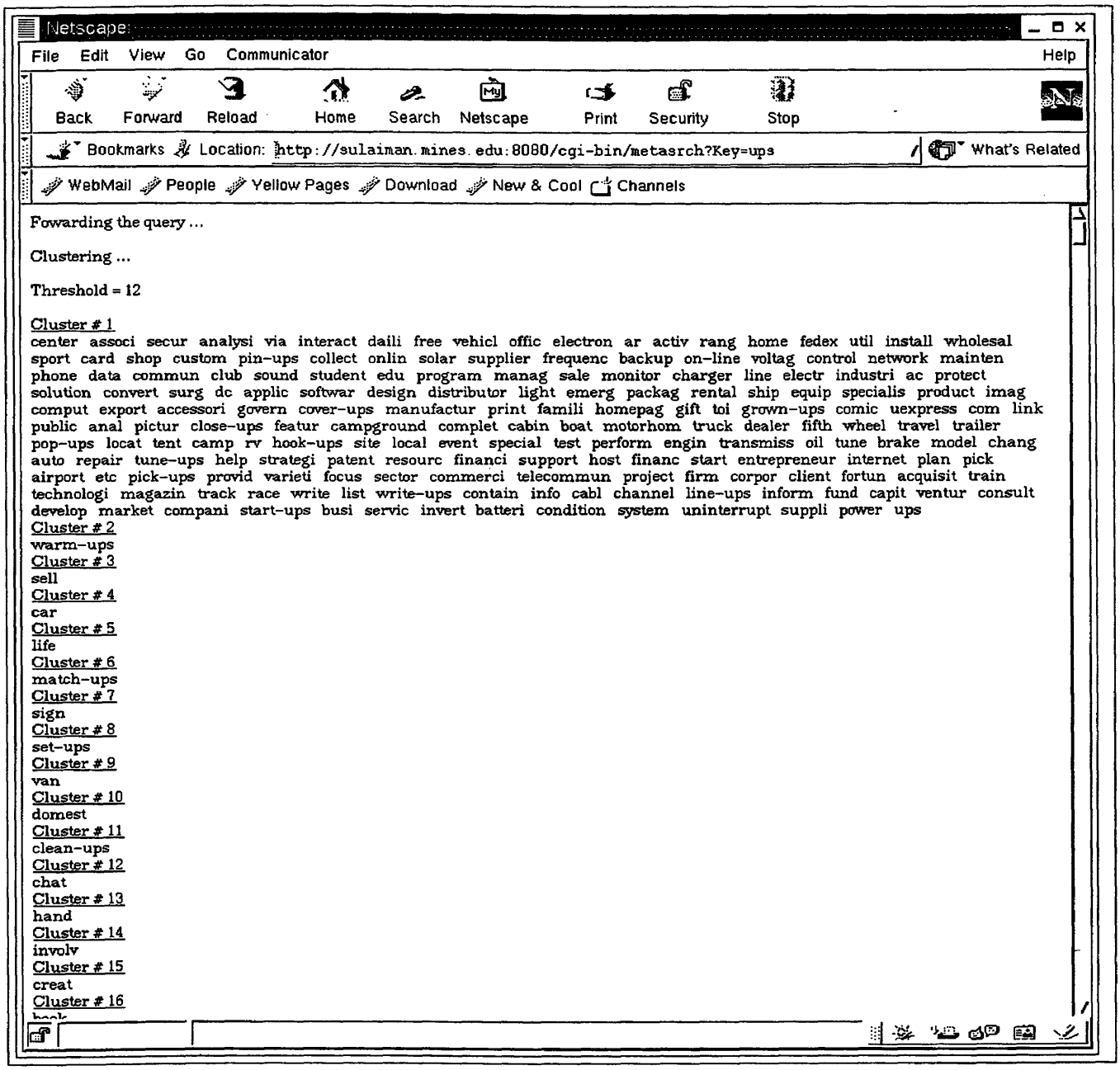


FIG. 4.4. Example keyword clusters generated by using AHC for query "UPS".

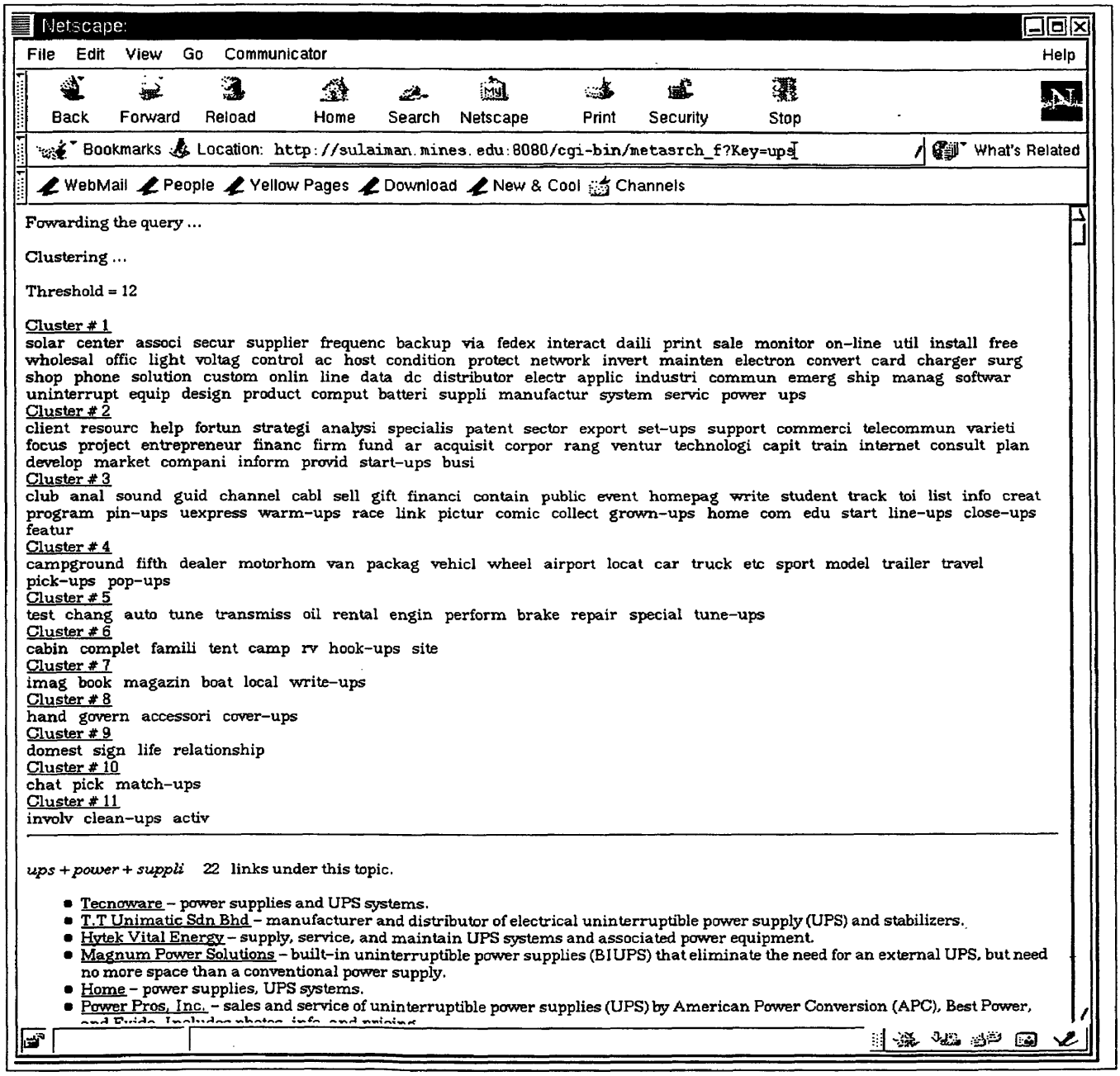


FIG. 4.5. Example keyword clusters generated by using CAFOG for query "UPS".

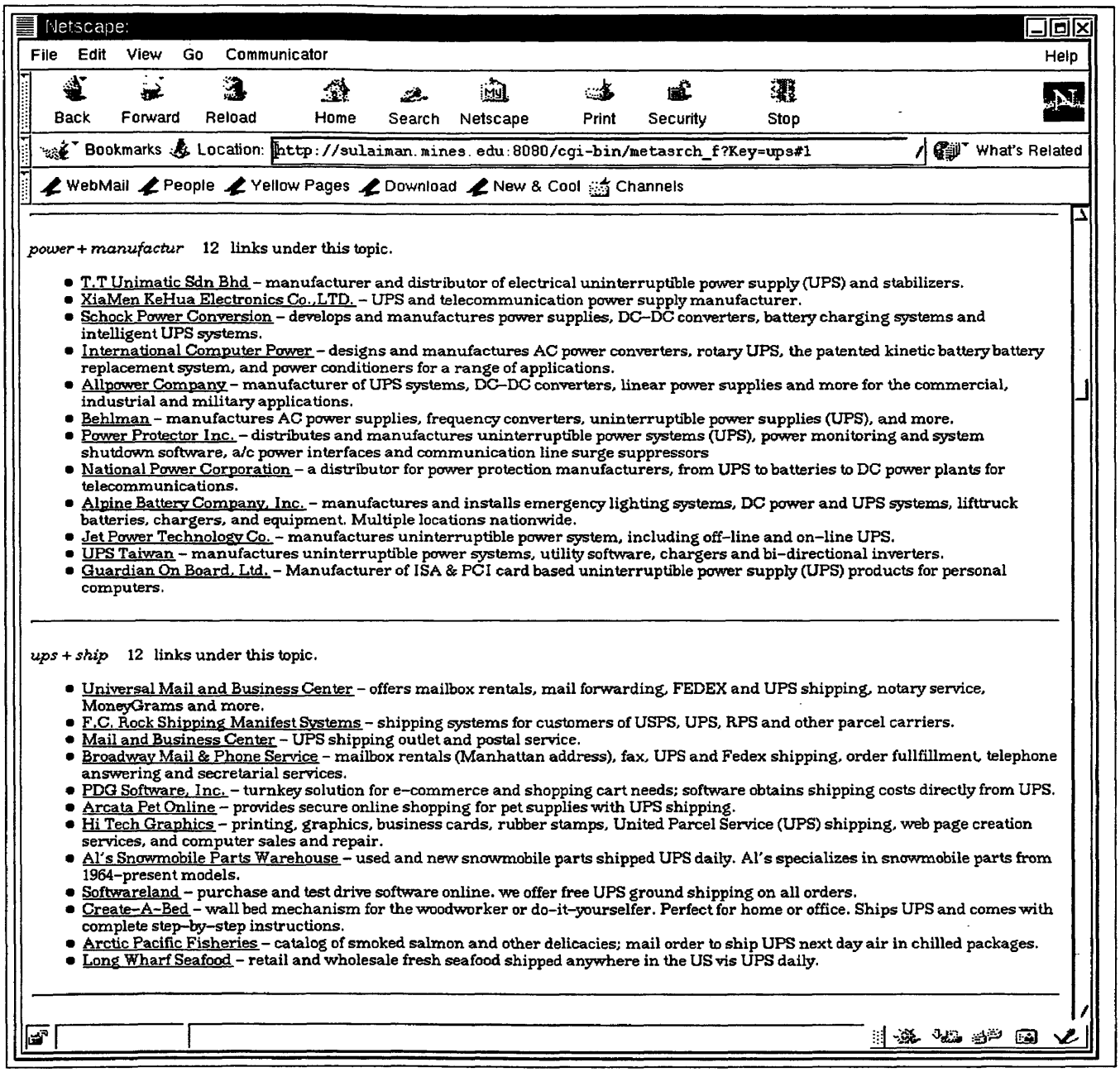


FIG. 4.6. Example snippet groups with subtopics for query "UPS".

Chapter 5

SUMMARY AND FUTURE WORK

5.1 Summary

5.1.1 Summary of the Proposed Clustering Algorithm

In this thesis, we presented a robust fuzzy relational clustering algorithm called Clustering by Agglomeration of Fuzzy Object Granules (CAFOG), which is especially suitable for dealing with large data sets. CAFOG consists of three sequential stages.

- *The first stage is an optional pre-clustering process called Leader Clustering or LC.* LC generates a set of n_L fine sub-granules, each represented by a leader. Each leader has a weight value that is equal to the number of objects it represents. The LC process keeps most of the useful information of the raw input while reducing the size of the data set for further processing. LC requires a heuristic threshold T_L. If T_L is too large, it may degrade the quality of the final results; if T_L is too small, we may not obtain the expected reduction of the input.

- *The second stage of CAFOG is Fuzzy c-Medoids Clustering or FCMdd.* FCMdd is the main body of CAFOG. It starts with a set of initial medoid candidates.

It takes only a few iterations to update the medoids and converges to a local minimum of the objective function. FCMdd can be made robust by using a trimming technique that identifies outliers (noise). The resulting algorithm is called RFCMdd. RFCMdd is fuzzy, so it can deal with complex overlapping data sets and generate high quality fuzzy object granules (subclusters). RFCMdd is also relational, which means that it does not require the input data set to be represented by feature vectors. However, RFCMdd does require the specification of the number of granules (subclusters), which is the same as the number n_M of medoids. We heuristically choose n_M to be 3 to 5 times the expected number of final clusters. In our application $n_M = 20$.

- *The third stage of CAFOG is Fuzzy Agglomerative Clustering or FAC.* FAC is derived from the well-known AHC [34] algorithm. It is a hierarchical clustering algorithm, and is used to cluster the output of (R)FCMdd, which is a set of granules (subclusters). FAC has the capability to deal with complex clusters because it incorporates the"overlap" between subclusters in the dissimilarity measure. FAC automatically determines the number c of final clusters.

The computational complexity of LC is $\mathcal{O}(n_L n_I)$, where n_L is the number of leaders and n_I is the number of input objects. As discussed in Sections 3.3 and 3.4, the complexity of RFCMdd (and its non-robust version FCMdd) is $\mathcal{O}(n_L^2 n_M)$, where n_L is the number of input objects or leaders and n_M is the number of medoids. There

is some extra computation for the initialization, which depends on the initialization method. Initialization I requires $\mathcal{O}(n_M)$ computations, and Initialization II and III require $\mathcal{O}(n_L n_M^2)$ computations. FAC works on the outputs of FCMdd or RFCMdd. Its complexity also depends on the dissimilarity measure used. As discussed in Section 3.5, the complexity of FAC when we use D_{ik}^o (i.e., dissimilarity modified by overlap) or D_{ik}^b (i.e., dissimilarity modified by bridge strength) is $\mathcal{O}(c^2 n_M)$, where c is the number of final clusters. The complexity of FAC when we use D_{ik}^g (i.e., dissimilarity modified by gap strength) is $\mathcal{O}(c^2 n_M \log n_M)$. If we apply the optional final pass refinement, then we have an additional complexity of $\mathcal{O}(n_I n_M)$.

5.1.2 Summary of the Proposed Application

In Chapter 3, we have illustrated the use of CAFOG in a document clustering application. The main application is the use of CAFOG in a search engine proxy to cluster keywords and generate concepts. The search engine proxy provides an interface which facilitates easy browsing of the query responses returned from a regular search engine (such as Yahoo). It presents the summaries of the Web links at two levels. The higher level is the concept-cluster level that consists of groups of keywords where each group represents a concept. The lower level is the subtopic level that consists of Web links assigned to each subtopic of a concept.

Our approach to clustering in our search engine proxy differs from other document or snippet clustering approaches by the fact that we cluster the keywords instead

of the snippets. Although the final goal is still to group the snippets, by clustering the keywords, we have two advantages:

1. The processing time of the keyword clustering is independent of the number of responses from the regular search engine. This leads to a total processing time complexity that is strictly linear in the number of tokens in the query response.

2. In the second level, we can guarantee that each summary assigned to a particular subtopic contains *all* keywords that characterize the subtopic. This fact admittedly (subjectively) improves the quality of the grouping.

The search engine proxy reads the response from Yahoo as a character stream, parses the HTML tags and tokenizes the stream. We use the Lex Tree to obtain the statistical information of the tokens. The memory size of the Lex Tree is upper-bounded by $\mathcal{O}(m_K n_S)$, where m_K is the number of salient keywords selected to represent the concepts and n_S is the number of summaries returned by the search engine in response to the query. The complexity of building the Lex Tree is $\mathcal{O}(n_T)$, where n_T denotes the total number of tokens included in all returned summaries. We use the Apriori algorithm to generate subtopics after we generate the concepts (represented by keyword clusters), which is quite efficient. Finally, the proposed method to assign each snippet to subtopics has a complexity of $\mathcal{O}(n_S n_{ST})$, where n_{ST} is the number of subtopics.

5.2 Future Work

The application we presented is only a portion of our on-going effort. There are still many aspects of the application that can be improved.

1. *More and Better Comparisons*

   We realize that the comparison we made between CAFOG and AHC in the search engine proxy application is somewhat subjective and limited. More objective measures for a comparison are necessary. Also, comparisons with other dissimilarity measures proposed in Section 3.5 could be made.

2. *Other Experiments*

   More experiments can be done in the search engine proxy application. For example, we can try using RFCMdd instead of the FCMdd to cluster the keywords. We can also explore assigning a weight to each keyword where weight is related to the IDF of the keyword or the numbers of snippets containing the keyword.

3. *More Applications*

   Our search engine proxy is only one of the applications of CAFOG. Hopefully we can find more significant applications for it, for example, categorization of users or Web documents.

4. *Larger Lexical Units*

   Recent research on natural language processing shows that single-word key terms are not the best way to describe the contents of Web objects. Other lexical units such as noun phrases may do better [2].

5. *More Dissimilarity Measures*

   We also realize that the dissimilarity measure used in FCMdd plays a very important role in the clustering algorithm. The dissimilarity measure should be data- and application-dependent. It heavily affects the quality of the clustering results. More research could be done for better dissimilarity measures.

6. *Concurrency and Parallelism*

   We may use concurrency and parallelism to improve the efficiency further. In the search engine proxy application, most of the procedures can use concurrency and parallelism. We could implement it as a multi-process/multi-thread pool that queries the regular search engine concurrently.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of SIGMOD '93*, pages 207–216, Washington D. C., 1993.

[2] P. G. Anick and S. Vaithyanathan. Exploiting clustering and phrases for context-based information retrieval. In *Proceedings of SIGIR '97*, pages 314–323, Montreal, 1997.

[3] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering from Heterogenous, Distributed Environments*, March 1995.

[4] P. Bajcsy and N. Ahuja. Location- and density-based hierarchical clustering using similarity analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:1011–1015, 1998.

[5] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, New York, 1981.

[6] C. Chang and C. Hsu. Enabling concept-based relevance feedback for information retrieval on the www. *IEEE Transcations on Knowledge and Data Engineering*, 11, 1999.

[7] J. Chen, A. Mikulcic, and D. H. Kraft. An integrated approach to information retrieval with fuzzy clustering and fuzzy inferencing. In O. Pons, M. Ampara Vila, and J. Kacprzyk, editors, *Knowledge Management in Fuzzy Databases.* Physica Verlag, Heidelberg, Germany, 1998.

[8] W. B. Croft. *Organizing and Searching Large Files of Documents.* PhD thesis, Cambridge University, 1978.

[9] D. Cutting, D. Krager, J. Pedersen, and J. Tukey. Scatter/gather: A cluster based approach to browsing large document collections. In *Proceedings of SIGIR '98*, pages 103–114, Montreal, June 1996.

[10] R. N. Davé and R. Krishnapuram. Robust clustering methods: A unified view. *IEEE Transactions on Fuzzy Systems*, 5(2):270–293, 1997.

[11] B. S. Everitt and Hand. *Finite Mixture Distributions.* Chapmand and Hall, London, England, 1981.

[12] K. S. Fu. *Syntactic Pattern Recognition and Applications.* Academic Press, San Diego, CA, 1982.

[13] K. C. Gowda and E. Diday. Symbolic clustering using a new similarity measure. *IEEE Transactions on Systems, Man, and Cybernetics*, 20:368–377, 1992.

[14] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient algorithm for large databases. In *Proceedings of SIGMOD '98*, pages 73–84, Seattle, June 1998.

[15] R. J. Hathaway and J. C. Bezdek. Switching regression models and fuzzy clustering. *IEEE Transactions on Fuzzy Systems*, 1(3):195–204, 1993.

[16] R. J. Hathaway and J. C. Bezdek. NERF c-means: Non-Euclidean relational fuzzy clustering. *Pattern Recognition*, 27:429–437, 1994.

[17] http://cswww.vuse.vanderbilt.edu/ dfisher/tech-reports/tr 88-05/node5.html.

[18] http://www.csa.com.

[19] http://www.firefly.com.

[20] P. Jarccard. *Nouvelles recherches sur la distribution floral.* Bull. Soc. Vard. Sci. Nat., Paris, 1908.

[21] A. Joshi, S. Weerawarana, and E. Houstis. On disconnected browsing of distributed information. In *Seventh IEEE Intl. Workshop on Research Issues in Data Engineering (RIDE)*, pages 101–108, 1997.

[22] L. Kaufman and P. J. Rousseeuw. Clustering by means of medoids. In Y. Dodge, editor, *Statistical Data Analysis Based on the L_1 Norm*, pages 405–416. North Holland/Elsevier, Amsterdam, 1987.

[23] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data, An Itroduction to Cluster Analysis.* John Wiley & Sons, Brussels, Belgium, 1990.

[24] J. Kim, R. Krishnapuram, and R. N. Davé. Application of the least trimmed squares technique to prototype-based clustering. *Pattern Recognition Letters*, 17:633–641, 1996.

[25] R. Krishnapuram, H.Frigui, and O. Nasraoui. Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation: Part i and part ii. *IEEE Transactions on Fuzzy Systems*, 3(1):29–43, 44–60, 1995.

[26] R. Krishnapuram and J. M. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2):98–110, 1993.

[27] O. Nasraoui, H.Frigui, A. Joshi, and R. Krishnapuram. Mining web access logs using relational competitive fuzzy clustering. In *Proceedings of NAFIPS'99*, New York, NY, June. 1999.

[28] O. Nasraoui and R. Krishnapuram. A genetic algorithm for robust clustering based on a fuzzy least median of squares criterion. In *Proceedings of NAFIPS'97*, pages 217–221, Syracuse, NY, Sept. 1997.

[29] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th VLDB Conference*, pages 144–155, Santiago, Chile, Sept. 1994.

[30] M. Pazzani, L. Nguyen, and S. Mantik. Learning from hotlists and coldlists:

Towards a www information filtering and seeking agent. In *Proceedings of Tools with Artificial Intelligence '95*, 1995.

[31] G. D. Ramkumar and A. Swami. Clustering data without distance functions. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 21:9–14, 1998.

[32] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* John Wiley and Sons, New York, 1987.

[33] T. A. Runkler and J. C. Bezdek. Ace: A tool for clustering and rule extraction. *IEEE Transactions on Fuzzy Systems*, 1999.

[34] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy - The Principles and Practice of Numerical Classification.* W. H. Freeman, San Francisco, CA, 1973.

[35] Y. El Sonbaty and M. A. Ismail. Fuzzy clustering for symbolic data. *IEEE Transactions on Fuzzy Systems*, 6:195–204, 1998.

[36] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proceedings of VLDB '95*, Zurich, 1995.

[37] L. Terveen, W. Hill, and B. Amento. Phokas - asystem for sharing recommendations. In *Comm. ACM 40:3*, 1997.

[38] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* Academic Press, San Diego, CA, 1999.

[39] Westby Thomas L Naps. *Introduction to Data Structures and Algorithm Analysis.* West Publishing Company, College & School Division, 1992.

[40] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of SIGIR'98*, Melbourne, Australia, Aug. 1998.

[41] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of SIGMOD '98*, pages 103–114, Montreal, June 1996.