A Hierarchical Monothetic Document Clustering Algorithm for Summarization and Browsing Search Results

Krishna Kummamuru
IBM India Research Lab
Block 1, IIT, Hauz Khas
New Delhi 110016 INDIA
kkummamu@in.ibm.com

Rohit Lotlikar
IBM India Research Lab
Block 1, IIT, Hauz Khas
New Delhi 110016 INDIA
rohitmlo@in.ibm.com

Shourya Roy
IBM India Research Lab
Block 1, IIT, Hauz Khas
New Delhi 110016 INDIA
rshourya@in.ibm.com

Karan Singal
Dept. of Computer Science and Engg.
IIT-Guwahati
North Guwahati 781039 INDIA
karan_iitg@yahoo.co.in

Raghu Krishnapuram
IBM India Research Lab
Block 1, IIT, Hauz Khas
New Delhi 110016 INDIA
kraghura@in.ibm.com

ABSTRACT

Organizing Web search results into a hierarchy of topics and subtopics facilitates browsing the collection and locating results of interest. In this paper, we propose a new hierarchical monothetic clustering algorithm to build a topic hierarchy for a collection of search results retrieved in response to a query. At every level of the hierarchy, the new algorithm progressively identifies topics in a way that maximizes the *coverage* while maintaining *distinctiveness* of the topics. We refer the proposed algorithm to as *DisCover*. Evaluating the quality of a topic hierarchy is a non-trivial task, the ultimate test being user judgment. We use several objective measures such as coverage and reach time for an empirical comparison of the proposed algorithm with two other monothetic clustering algorithms to demonstrate its superiority. Even though our algorithm is slightly more computationally intensive than one of the algorithms, it generates better hierarchies. Our user studies also show that the proposed algorithm is superior to the other algorithms as a summarizing and browsing tool.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – Clustering

General Terms

Algorithms

Keywords

Automatic Taxonomy Generation, Clustering, Summarization, Data Mining, Search

1. INTRODUCTION

The lack of a central structure and freedom from a strict syntax is responsible for making a vast amount of information available on the Web, but retrieving this information is not easy. Ranked lists returned by search engines are still a popular way of searching and browsing the Web today. However, this method is highly inefficient since the number of retrieved search results can be in the thousands for a typical query. Most users just view the top ten results and therefore might miss relevant information. Moreover, the criteria used for ranking may not reflect the needs of the user. A majority of the queries tend to be short [3], thus making them non-specific or imprecise [19]. The inherent ambiguity in interpreting a word or a phrase in the absence of its context means that a large percentage of the returned results can be irrelevant to the user.

Ranked lists have been found to be fairly effective for *navigational tasks* such as finding the URL of an organization. However, since the results are not summarized in terms of topics, they are not well suited for browsing tasks. One possible solution is to create a static hierarchical categorization of the entire Web and use these categories to organize the search results of a particular query. For example, the *dmoz* (*www.dmoz.org*) directory that categorizes Web sites is manually created by about 52 thousand editors. However, this solution is feasible only for small collections. For example, *dmoz* covers less than 5% of the Web. Secondly, even if we were to categorize the entire Web either manually or automatically, the categories may not be useful in organizing the search results of a particular query [18, 6]. This is so because some or all of the sub-topics associated with the user query may not be present in the directory.

It has been observed that post-retrieval document clustering typically produces superior results [8]. Taxonomies are generated using document clustering algorithms which typically result in topic or concept hierarchies. Concept hierarchies expose the different concepts present in the document (or search result) collection, as top level nodes in the hierarchy and the user can choose the concept he is interested in and can browse it in detail. Examples of search engines that return search results in terms of a hierarchy include *vivisimo* (*www.vivisimo.com*) and *kartoo* (*www.kartoo.com*).

Automatic taxonomy generation (ATG) algorithms can be categorized into two classes based on how the documents are assigned to different clusters. *Monothetic* algorithms are those in which a document is assigned to a cluster based on a single feature, whereas *polythetic* algorithms assign documents to the clusters based on

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

multiple features. Monothetic clustering is well suited for generating hierarchies for summarization and browsing search results because each cluster is described by a single feature or a concept and all the documents present in a cluster contain this feature. Hence, the user can easily understand clusters generated by monothetic clustering algorithms.

In this paper, we propose a novel algorithm for post-retrieval hierarchical monothetic clustering of search results to generate concept hierarchies. As the algorithm progressively identifies clusters it tries to maximize the distinctiveness of the monothetic features describing the clusters while at the same time maximizing the number of documents that can be described or covered by the monothetic features. We refer to the proposed algorithm as *DisCover*. One of the challenges we address in this paper is that of evaluating the performance of the algorithms that generate concept hierarchies. We compare the performance of DisCover with that of two of other known monothetic algorithms. The two algorithms are CAARD and DSP and they will be explained in the next section. This comparison is based on certain objective measures and it shows that DisCover results in hierarchies with superior coverage and reach time (explained later). DisCover takes slightly more time (19ms) than CAARD to generate hierarchies, but it takes much less time than DSP. In addition to comparison based on objective measures, we have also conducted user studies evaluate the performance of the algorithms subjectively. The user studies reveal that the hierarchies obtained using DisCover are more meaningful than to those obtained by CAARD and DSP.

In Section 2, we describe some of the earlier work on taxonomy generation. In Section 3, we present the proposed DisCover algorithm. In Section 4, we describe the experimental setup that was used for conducting the experiments. We will also describe the various heuristics that we used for identifying interesting phrases (i.e., candidates for monothetic features) in the document collection in detail. We present the comparisons in Section 5 and report the results from our user studies in Section 6. We summarize our contributions and propose some possible extensions in the last section.

2. RELATED WORK

2.1 Automatic Concept Hierarchies

As mentioned in the previous section, some of the ATG algorithms are *monothetic* in the sense that a document is assigned to a cluster based on a single feature, while the others are *polythetic*. Popular examples of polythetic document clustering algorithms are K-means [1] and agglomerative algorithms [22]. In these algorithms, each cluster is described by several words or phrases. The grouping of documents can be done by clustering documents or words or by simultaneously clustering both documents and words. The last approach is called *co-clustering*. ATG algorithms can be classified as either *top-down* or *bottom-up*, depending on how they build the hierarchy. A survey of various ATG techniques can be found in [9]. We briefly describe some of the techniques below.

Zamir and Etzioni [26] present an interface (called *Grouper*) to the HuskySearch meta search engine [21] that dynamically groups the search results into clusters labeled by phrases extracted from snippets. Grouper uses an algorithm called Suffix Tree Clustering (STC) for forming groups of "snippets" (or summaries) of Web pages. STC is based on standard techniques from the literature that allow the construction of "suffix trees" in time that is linear in the number of document snippets, assuming that the number of words in each snippet is bounded by a constant.

Vaithyanathan and Dom [23, 24] define a model for flat (i.e., one-level) clustering, and then generalize it to generate hierarchical clusters. The model assumes that the feature set T can be partitioned into two conditionally independent sets: a 'useful' set U, and a 'noise' set N. They extend the noise/useful concept to hierarchical clustering by interpreting the 'noise' set associated with a particular node to be the set of features that have a common distribution over child nodes of the given the node, and the 'useful' set to be the set of features that can be used to discriminate among the child nodes.

There have been some attempts to generate hierarchies using the thesaural relationship between words and phrases.They analyze the context (phrase) in which a term occurs to infer the relationships between various terms [5, 7].

One of the early works in monothetic clustering is the subsumption algorithm by Sanderson and Croft [20]. The subsumption algorithm builds concept hierarchies by finding pairs of concepts (x, y) in which x subsumes y. The algorithm computes the subsumption relationships between some selected pairs of words and phrases (x, y) from the document collection and then retains only those pairs in which x subsumes y. The hierarchy is then built in a bottom-up fashion.

Lawrie et al. [13] consider an approach based on a set of topic terms and a set of vocabulary terms to generate concept hierarchies. They propose a language model composed of all the conditional probabilities $\Pr_x(A|B)$ where $\Pr_x(A|B)$ is the probability of occurrence of A in the x-neighborhood of B. Here, A is a topic term and B is a vocabulary term. In the experiments reported in [13], the topic terms and the vocabulary terms are the same and are those that occur in at least two documents excluding numbers and stopwords. The ultimate goal is to find the set of terms that have maximal predictive power and coverage of the vocabulary. To achieve this goal, the language model is converted into a graph and the problem is posed as Dominating Set Problem (DSP). Since DSP is known to be NP complete, the authors propose a greedy approximation to solve DSP. The solution provides a set of terms that can be used as labels at the top level. To find the sub-topics of a top-level topic, the language model is constructed on the terms occurring in the neighborhood of the topic and the corresponding DSP is solved. The procedure can be applied recursively to obtain a hierarchy of topics. A variation of this method has been applied to Web searches [14].

Kummamuru and Krishnapuram [11] consider the inclusion relation between while generating concept hierarchies. A set of words and phrases representing concepts is first identified from the corpus. The degree to which the meaning of one concept w_i in another concept w_j is estimated from the document corpus as $|\mathrm{w}_j \cdot \mathrm{w}_i|/|\mathrm{w}_i|$ where w is a n dimensional vector in which the i-th element is 1 if concept w is present in i-th document and is zero otherwise. The goal is to find a minimal subset of concepts whose elements are as distinct from each other as possible and each concept not in the subset has an inclusion degree in at least one of concepts in the subset that is greater than a predefined threshold, η. The authors propose a fast greedy algorithm called Clustering Algorithm for Asymmetrically Related Data (CAARD) to find the solution to the problem. The concept hierarchy is built by recursively applying CAARD to the concepts in each cluster until a terminating condition is reached. CAARD adaptively finds η at various levels so that the algorithm results in a hierarchy of a pre-specified size.

There is another class of approaches in which keywords are clustered instead of documents. In this approach, two keywords are considered similar if they occur in the same set of documents. This type of clustering is known as distributional clustering [16]. Fuzzy Co-clustering of Documents and Keywords (FCoDoK) [10], Fuzzy

Simultaneous KeyWord Identification and Clustering of Documents (FSKWIC) [4], and Rowset Partitioning and Submatrix Agglomeration (RPSA) [15] are examples of co-clustering.

2.2 Evaluation of Taxonomies

Evaluation of the quality of taxonomies generated by a particular algorithm is an important and non-trivial task. We briefly review some of the relevant evaluation measures used in the literature. In [26], Zamir and Etzioni manually determine the precision of the clustering algorithm. In [27], Zhao and Karypis use the *F-Score* to evaluate the accuracy with which the documents are assigned to the clusters. However, this approach requires the use of ground truth, which is unknown for collections of documents returned by a search engine. Sanderson [19] performed a user study to evaluate the quality of the relationship between a given concept and its child and parent concepts. In [12] (a longer version of [14]), Lawrie and Croft used three different evaluation measures. The *summary evaluation* compares the Expected Mutual Information Measure (EMIM) of terms in the hierarchy with the EMIM of the top TF-IDF terms. This measure only estimates the goodness of concepts when compared to the top TF-IDF terms and does not necessarily reflect the end user requirements. The *reachability* criterion measures the ability to find documents within the hierarchy. The *reach time* criterion measures the time taken to find a relevant document.

3. MONOTHETIC DOCUMENT CLUSTERING ALGORITHM

3.1 Desirable Properties of Taxonomies

The main purpose of building a taxonomy is to provide a meaningful navigational and browsing mechanism by organizing large amounts of information into a small number of hierarchical clusters [27]. In particular, the taxonomy should provide an overview of the various concepts present in the collection of documents, and reduce the search time associated with locating documents of interest.

In this section, we enumerate some desirable properties of taxonomies generated from a corpus of documents. The proposed algorithm uses some of these desirable properties. In Section 5, we evaluate how well the taxonomies generated by the proposed algorithm satisfy these desirable properties.

Property 1: *Document Coverage.* Ideally, all the documents in the collection should be covered by the taxonomy. A document is said to be covered by a taxonomy if it lies in (or assigned to) at least one of the clusters in the top level of the taxonomy. This requirement means that the top level clusters should be chosen in such a way that they cover most of the documents in the corpus. A taxonomy generation algorithm that covers more number of documents will be considered to be better.

Property 2: *Compactness.* Since the main purpose of the taxonomy generation is to summarize and provide a better browsing experience, the taxonomy should be as compact as possible. If a taxonomy becomes too wide or too deep, then the basic purpose of taxonomy generation may not be served. A ranked list (which can be considered as one-level deep taxonomy) may have a good coverage but it is not compact.

Property 3: *Sibling Node*[1] *Distinctiveness.* Each of the nodes, especially those at the top level in the taxonomy, represents a concept[2] present in the search results. At any level of the hierarchy, the sibling concepts should be as different as possible from each other to minimize ambiguity while browsing the documents in the hierarchy.

Property 4: *Node Label Predictiveness.* A good taxonomy should help the user find documents of interest with minimum effort. The labels of the node guide the user in locating a document in the taxonomy. Hence, the node labels should be chosen such that they are good indicators of the documents they contain.

Property 5: *Reach time.* The average time it takes to locate search results of interest is also an important criterion for taxonomies. Ideally, we should be quickly able to locate search results of interest within the hierarchy. *Reach time* is quantified more formally in Section 5.

Property 6: *General to specific.* The node labels in the hierarchy are actually the concepts associated with the query. In the hierarchy, the most general concepts should be associated with the root, and the node label of any node should be more specific (less general) than that of its parent and at the same time it should be less specific (more general) than those of its children. The generality or specificity of the node labels needs to be considered within the context of the query.

Among the above desirable properties, we use Properties 1, 2 and 3 for the development of the algorithm. In Section 5, we evaluate to what extent Properties 1, 2 and 5 are satisfied by the taxonomies obtained by DisCover, CAARD and DSP. We also compare the computational complexities of the three algorithms. Since Properties 4 and 6 are difficult to quantify, we evaluate them subjectively along with other measures in Section 6.

It may be noted that compactness and document coverage could be contradicting requirements. Coverage can be increased by adding more top level nodes. However, when the compactness criterion is added, the algorithm would be first forced to capture those concepts that are present in a large portion of the document collection.

3.2 DisCover Algorithm

We assume that each document in the collection can be represented by a set of concepts. Each node in the hierarchy is associated with a concept (i.e., the node label) and all documents under that node contain that concept. In general, each of these documents will contain several other concepts as well. We refer to the union of all these concepts as *concepts under the node*. Monothetic clustering of the documents under a node involves selecting a subset of those concepts, optimal in some sense, and associating a child node with each of them.

For ease of browsing and speed of execution, it is desirable to compute a compact hierarchy that initially has a limited number of child nodes under each node, but provide the user, via the user interface, the ability to progressively increase the number of child nodes of any node of interest. However, addition of new child nodes should not require re-clustering of documents under the node. Otherwise the speed will be compromised. Therefore, the child nodes need to be generated in a particular sequence that is optimal in some sense. For these reasons, we formulate our algorithm as that of sequentially selecting concepts from a set of concepts in an optimal manner. This is equivalent to saying that we wish to find an optimal ordering or permutation of the set of concepts. Specifically,

[1]We use the word 'node' to refer to a 'cluster' in a taxonomy.

[2]We use the word *concept* to also represent either a word or a phrase.

we would like that the next concept selected be the one that will increase the coverage maximally (Property 1) and will be maximally distinct from its existing siblings (Property 3). By providing ability to grow the hierarchy progressively, we allow users to choose upon a suitable trade off between coverage and compactness (Property 2) that is appropriate for the task at hand.

Let $C = \{c_1, \ldots, c_n\}$ be the complete set of ordered concepts under a node in the taxonomy. Let $C(\alpha) = \{c_{\alpha(1)}, c_{\alpha(2)}, \ldots c_{\alpha(n)}\}$, where α is a permutation on $\{1, \ldots, n\}$, denote a permuted sequence of concepts. Let $S_{\alpha,k-1} = \{c_{\alpha(1)}, \ldots, c_{\alpha(k-1)}\}$ represent $(k-1)$ (where $((1 \leq k \leq n))$) concepts that have already been chosen sequentially. Let $U_{\alpha,k-1} = C - S_{\alpha,k-1}$ denote the remaining concepts in C that have not entered the sequence. We need to pick the next concept from $U_{\alpha,k-1}$ add to $S_{\alpha,k-1}$, or equivalently we need to determine $\alpha(k)$. The optimality criterion to be maximized for determining $\alpha(k)$ is

$$\alpha(k) = \arg\max_j g(S_{\alpha,k-1}, c_j) \quad c_j \in U_{\alpha,k-1}, \tag{1}$$

where, $g(s_{\alpha,k-1}, c_j)$ is defined as follows:

$$g(S_{\alpha,k-1}, c_j) \triangleq w_1 g_c(S_{\alpha,k-1}, c_j) + w_2 g_d(S_{\alpha,k-1}, c_j). \tag{2}$$

Here, g_c and g_d are functions that reflect document coverage and sibling distinctiveness respectively, and w_1 and w_2 are the weights associated with the respective factors.

Let $d(C)$ denote the set of all documents covered by the concept C. We define

$$g_c(S_{\alpha,k-1}, c_j) \triangleq |d(c_j) - d(S_{\alpha,k-1})|. \tag{3}$$

The quantity on the right is the number of documents in $d(c_j)$ that are not in $d(S_{\alpha,k-1})$. In other words $g_c(.)$ quantifies the increase in coverage that would result by addition of c_j as a concept node.

To compute the distinctiveness of a node c from a set of nodes S, we consider of the concepts covered under S and c or concepts that co-occur along with the concepts S and c. Let $t(S)$ and $t(c)$ denote the sets of concepts under S and c respectively. Then

$$g_d(S_{\alpha,k-1}, c_j) \triangleq |t(c_j) - t(S_{\alpha,k-1})|. \tag{4}$$

In other words $g_d(.)$ quantifies the increase in the total number of concepts when c_j is added to $S_{\alpha,k-1}$.

To determine $\alpha(k)$, we need to evaluate the RHS of (1), i.e., evaluate $g(S_{\alpha,k-1}, c_j)$, for all $c_j \in U_{\alpha,k-1}$. The entire process for computing the sequence $S_{\alpha,k}$ is described in the pseudo code in Figure 1. We refer to the proposed algorithm as the concept Distinctiveness and document Coverage (*DisCover*, for short) algorithm.

There are some similarities between the above formulation and that of traditional leader clustering [2]. Suppose we require to cluster the concepts into k clusters. Then, the first k concepts $c_{\alpha(1)}, \ldots, c_{\alpha(k)}$ represent the leaders of k clusters. The rest of the elements in $C(\alpha)$ can be assigned to one these k clusters that minimizes its dissimilarity (g) with the leader of the cluster. Note that a similarity threshold, η, is used in leader clustering algorithms as well as CAARD. Increasing the threshold increases the number clusters. Moreover, the cluster leaders result with a smaller value of η remain as cluster leaders when the data is reclustered with a higher value of η. This observation also motivated us to the above formulation.

Here, we formulated the problem of taxonomy generation as the problem of finding a permutation that optimizes an objective function. This is done having in mind the experience of the end-user of the system. Typically, a user browsing through a taxonomy would like to look at the sub-concepts at various levels of granularity. Large number of sub-concepts (clusters) are needed by a person that wants to see more detailed sub-concepts. Alternatively, as will be explained in the experimental setup, we can gradually show the concepts in $C(\alpha)$ to the user as she seeks more and more information.

The values of the weights w_1 and w_2 need to be determined empirically. We have chosen $w_1 = 0.8$ and $w_2 = 0.2$ by trial and error. In our experience, the concepts that are picked initially are the ones that are present in a large number of documents. The "niche" concepts begin to appear in later stages.

$S_{\alpha,0} = \emptyset$
for $k = 1$ to n {
  $\alpha(k) = \arg\max_j g(S_{\alpha,k-1}, c_j)$
  $S_{\alpha,k} = S_{\alpha,k-1} \cup c_{\alpha(k)}$
}

Figure 1: DisCover Algorithm - outer loop

popular queries	2 fast 2 furious, miss universe, sars, bugbear, david beckham
ambiguous queries	jaguar, latex, panther, java, blues

Table 1: Queries used in the user study.

4. EXPERIMENTAL SETUP

4.1 Introduction

In this section, we describe the system that we built and used for empirical evaluations and user studies. We selected the queries shown in Table 1. Of the ten queries, five are what we call *ambiguous* queries and the remaining five are *popular*. Ambiguous queries have multiple interpretations associated with them, whereas popular queries are some of the top gaining queries reported by Google in the month of June 2003. For each query, we requested 1000 (English only) search results from Google (http://www.google.com). The actual number of search results returned by Google for these ten queries varied from 564 to 910 with a mean of 818. Also, some non-English (generally French or Spanish results) would creep in. For each search result, we retrieve the title and snippet and converted them to plain text. Next, the *feature extraction* module (described in detail Section 4.2) builds an index for the collection of snippets. The feature extraction phase is common to all three algorithms. Finally, each algorithm accesses the index to generate its hierarchy. We have developed an interface, which will be explained below, that simultaneously displays the three hierarchies generated by CAARD, DisCover and DSP respectively. We set the parameters of all three algorithms to only display the top five nodes under any parent node. The purpose of doing so is to make it simpler for users to compare the hierarchies. We first explain the feature extraction module in the next section and the user interface in Section 4.3.

4.2 Feature Extraction

The purpose of feature extraction is to build an index of the various terms that occur in the collection. Once the index has been created, the original documents are no longer required. The index is a compact representation of the document collection. Not all of

the terms in the document collection should be included in the index. Common words and frequent words such as 'the', 'of', 'to' are poor predictors of the content of a document and have little value as indexing terms. Also, it is preferable that morphologically and semantically related terms are conflated into a single index term. For example, the terms "professor", "professorship", and "A professor" should be conflated into a single term such as "professor".

Nouns, adjectives and noun phrases are valuable as index terms since they are good indicators of the content of a document. Part-of-speech determination and extraction of noun phrases were done using a tool developed by IBM T. J. Watson Research Lab, which is in essence similar to [25].

The index is binary, it indicates whether a term occurs or does not occur in a document. Since we deal with short (1-3 line) snippets, it does not make sense to consider a term more important if it occurs multiple times in a document. Though feature extraction is not the focus of this work, good feature extraction is an important prerequisite for a good hierarchy. The following are the steps used for this purpose.

Step 1: *Term extraction.* Generally noun phrases, and single words that are either nouns or adjectives are most predictive of the content of the document. We identify these terms while ignoring other terms. We also ignore any single words less than 3 characters in length.

Step 2: *Adding constituent words and sub-phrases*. For each extracted phrase, we add constituent words of the phrase as well as sub-phrases to the list of concepts.

Step 3: *Stop-word elimination*. We eliminate words commonly found in Web documents (such as "page", "site", and "click") that are considered as *stopwords*.

Step 4: *Morphological generalization*. We normalize noun phrases by removing stopwords, determiners and prepositions. Single words and words part of noun phrases are stemmed using Porter's stemmer [17]. All terms are converted to lower-case.

Step 5: *Index creation*. An index is created for the terms resulting from Steps 1 to 4. The index is further refined by removing the terms that occur in less than two percent of the documents because they are unlikely to impact the hierarchy.

Step 6: *Generating node labels and thresholding*. We have represented concepts by stemmed words and phrases. However, these are not usually very meaningful for use as node labels. Therefore, we replace each stemmed term by the most frequently occurring original term.

4.3 User Interface

The user interface consists of a browser window with 4 frames as shown in Figure 2. This figure shows the GUI for the query "Latex". The three hierarchies in the left 3 frames from left to right correspond to CAARD, DisCover and DSP respectively. We hide the names of the hierarchies and call them Algorithm 1, 2 and 3 to obtain unbiased feedback from the users. When any node label (for any of the 3 hierarchies, at any level of the hierarchy) is clicked, the snippets contained in that node are displayed in the right most frame. Thus the user can see the documents assigned to any node of the hierarchies. A '$+$' sign inside a node indicates that the node may be expanded to show the lower-level nodes. When this sign is clicked the node is expanded to display the child nodes, and simultaneously the sign changes to '$-$' indicating that the child nodes may be collapsed. If a node contains more than 5 children, we display the *more* and *less* hyperlinks at the bottom. The user can click on them to see either more number of sub-concepts or less number of sub-concepts. However, for the user study, in order to make it easier for the hierarchies to be compared, we displayed only upto a maximum of 5 children for each node. Also we do not display nodes that contain fewer than 4 documents. Documents that do not fall into any of the displayed nodes are relegated to an "Others" node. In Figure 2, we have expanded one of the nodes ("rubber") in all the three hierarchies.

	DisCover	CAARD	DSP
CPU time (ms)	99.2	80.8	421.7
Complexity	$O(kn)$	$O(kn)$	$O(kn^2)$

Table 2: Average CPU time of algorithms in milliseconds.

5. EMPIRICAL EVALUATION

5.1 Computation Complexity

Let n be the number of concepts and k be the number of clusters that we are interested in generating. Then, the `for` loop in Figure 1 needs to be executed only k times instead of n times. Hence, the complexity of DisCover is $O(kn)$.

CAARD visits concepts in the decreasing order of their document frequencies and computes their inclusion with the already selected leaders. Hence, Card's complexity is also $O(kn)$. As explained in Section 2, DSP builds a language model. It computes conditional probabilities between every pair of concepts. This makes the complexity of DSP $O(kn^2)$.

In our experiments, we have also measured the execution time of all three algorithms. Table 2 shows the average CPU time of the algorithms for generating the hierarchies for the queries in Table 1 along with their computational complexity. These results reflect the above observations. However, DisCover takes a little longer than CAARD. This is mainly because DisCover computes concept distinctiveness in addition to document coverage. CAARD effectively computes document coverage only.

5.2 Coverage and Compactness

As mentioned in the Section 3, there is an implicit trade off between coverage and compactness. We analyze this behavior for the taxonomies obtained by the three algorithms as we consider more and more nodes. We compute the percentage of documents that has been covered. In the case when the top level node contains a child node with the query term as its label, we consider the sub-concepts of this node.

DSP uses a threshold that is computed using conditional probabilities. Therefore, the number of nodes returned by DSP dependents on the contents of the node in the taxonomy. Since DisCover and CAARD can generate as many nodes as required in the comparisons, we consider only the number of nodes generated by the DSP. Figure 3 shows the graph of coverage versus the number of nodes for two of the queries. Each of these graphs shows the behavior of all the three algorithms. As expected from the formulations, DisCover algorithm exhibits higher coverage than the other algorithms for the same number of nodes. Even though the behavior of DSP is quite similar to that of DisCover for small number of nodes, its coverage reduces below that of CAARD algorithm for higher number of nodes.

5.3 Reach Time

To quantify the *reach time* for a search result, we need to first outline an operational procedure to locate a search result of interest and then base the reach time upon that procedure. This procedure should mimic the procedure that a typical user of the system will follow.

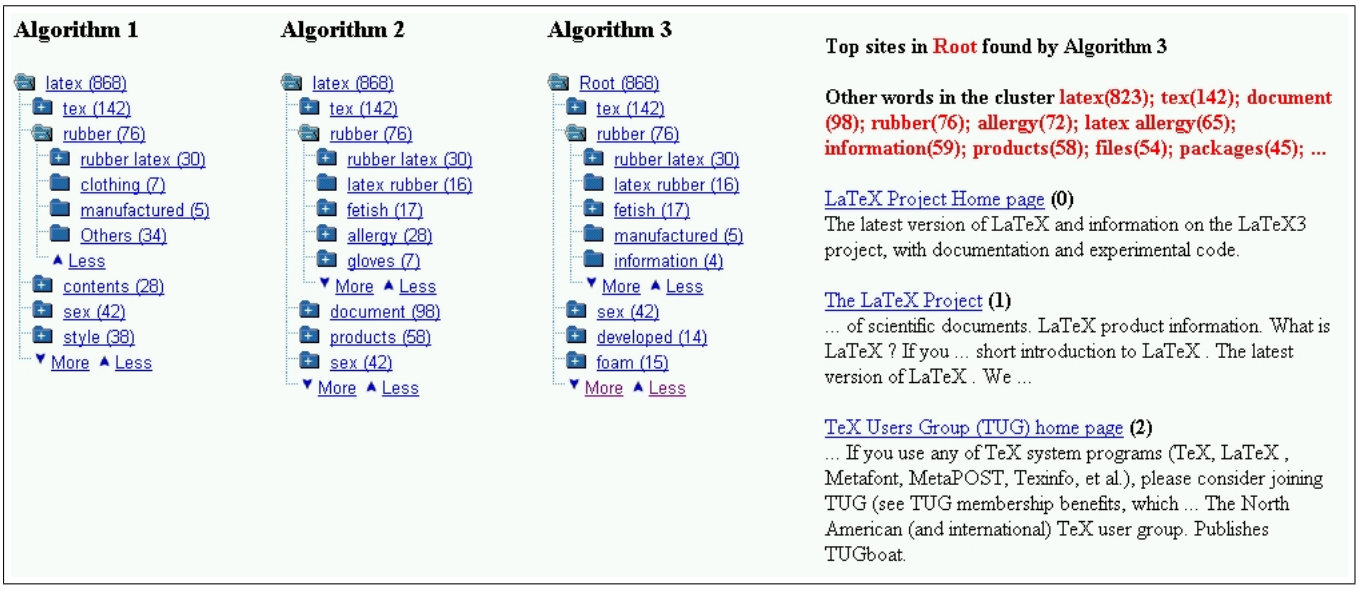


Figure 2: The user interface.

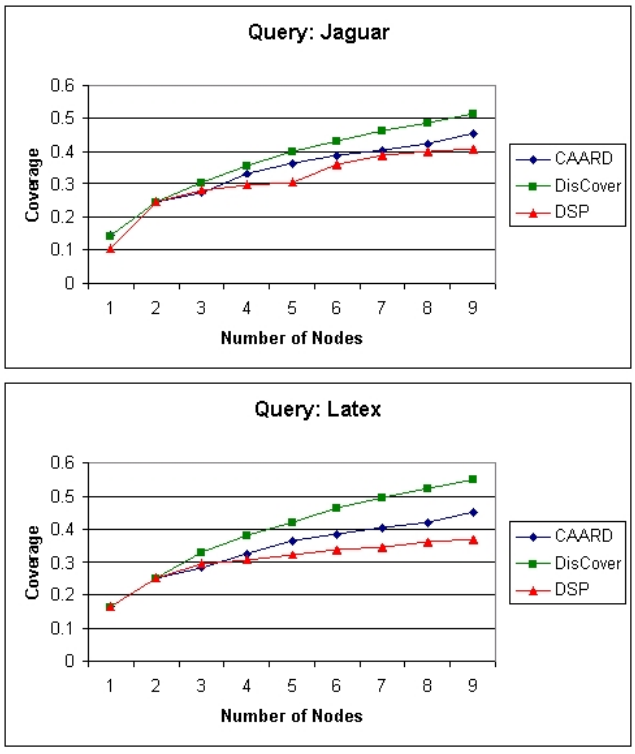


Figure 3: Effect of number of clusters (nodes) on coverage.

A typical user would inspect all the top level node labels and select a node which he or she feels is most likely to contain the search result(s) of interest. This takes a time proportional to the number of top level nodes, lets say it takes a time $\theta \times n_0$, where n_0 is the number of root level nodes and θ is a proportionality constant. The user will expand the selected node to display it's child nodes. Similarly, for the consecutive levels, user will inspect the node labels and select one that he/she feels is most likely to contain relevant search results. The time required for this selection can be similarly computed. The user would continue this process and drill down the hierarchy until a leaf node is reached. By adding up the times spent at each level, we obtain the path time t_{path} for this node. At this point, the user would scan the search results under that leaf node sequentially until the desired search result is found. If the desired snippet is located at the p^{th} position, then the reach time for the snippet is given by

$$t_{reach} = (\theta_s \times p) + t_{path}(q). \quad (5)$$

If a snippet is in multiple leaf nodes, we use the least of the reach times as the reach time for that snippet. To compute the average reach time over all snippets, we average the reach time over all the snippets. Snippets that do not fall into any node of the hierarchy, are assumed to lie in the "Others" node at the top-level of the hierarchy. For a ranked list the reach time for the i^{th} snippet is simply i, so the average reach time for a ranked list of N search snippets is

$$t_{Rankedlist} = 1/N \sum_{i=0}^{N-1} i = N/2. \quad (6)$$

Table 3 summarizes the reach time computed for the 2 sets of queries. Each of the numbers shown in the table is the average of reach times of hierarchies obtained using the queries from the corresponding query group. It may be observed from Table 3 that DisCover resulted in hierarchies with less reach times than the other algorithms.

	CAARD	DisCover	DSP	Ranked List
ambiguous	196	161	181	424
popular	130	87	221	377

Table 3: Comparison of the average reach time for different algorithms.

Question number	Criterion Evaluated
1a	Summarization by top level nodes
1b	Missing concepts in top level
1c	Redundancy among top level nodes
2a	Summarization by second level nodes
2b	Redundancy among second level nodes
3	Overall utility for browsing and searching

Table 4: Questions used in the survey.

6. USER STUDY

Comparing and evaluating hierarchies is not easy as many of the criteria are rather subjective in nature. The best way to evaluate the performance on such criteria is by performing user studies. This demonstrates whether the hierarchies generated are actually helpful to real users for browsing a particular document collection.

We performed the study on 17 volunteers who were both technical and non-technical employees of the IBM India Research Lab. The queries were assigned to each volunteer such that each volunteer evaluated 3 queries (except for one volunteer who evaluated 2 queries). This gave us 5 responses to each query, with a total of 25 responses for ambiguous queries and 25 responses for popular queries. Users were provided with a very short introduction to ATG and the motivation behind creating a taxonomy from a document collection. The users were not told which hierarchy corresponded to which algorithm.

Each user was asked to fill up an online questionnaire for each query. The questionnaire contained 6 questions. For each question, the user was asked to rate each hierarchy on a scale of 1-10. The 6 questions can be broadly divided into 3 groups. The first group of 3 questions pertain to the top level nodes of the hierarchy. The second group of 2 questions pertain to the second level nodes. In the final question, the user was requested to give an overall rating to the hierarchy. The criteria evaluated by the questions are shown in Table 4.

6.1 Analysis

Since the evaluation process is subjective, it is not meaningful to compare the actual scores between one user and another or between one query and another. However, it is meaningful to compare scores within a response. Thus, for each question in each of the 50 responses, we noted whether DisCover was rated "better than", "worse than" or "equal to" CAARD and DSP respectively. The responses for popular and ambiguous queries were separated out. The number of instances of each type was counted. These results are shown in Table 5 and 6, for ambiguous and popular queries respectively.

From Tables 5 and 6, it can be seen than DisCover is superior to DSP in almost every respect. In comparison with CAARD, DisCover is better in terms of summarization for both popular and ambiguous queries. However, in the redundancy aspect, for ambiguous queries, DisCover performs comparably to CAARD. For popular queries, CAARD outperforms DisCover in the redundancy aspect. The response to the last question *"Overall, which hierarchy is the best in terms of summarizing the search results and ease of browsing the results?"* indicates the overall sentiment of users toward the algorithms. The results show that DisCover outperforms both CAARD and DSP. In particular, for ambiguous queries the difference is higher. This is as expected because, since the purpose of hierarchies is to identify the different concepts in the document collection, the utility of a good hierarchy is more evident when the queries are ambiguous. An alternate view of the results for question 3 is shown in Figure 4. DisCover is ranked first far more often than it is ranked second or third when compared with CAARD or DSP, particularly in the case ambiguous queries.

Question number	relative to CAARD			relative to DSP		
	better	equal	worse	better	equal	worse
1a	14	6	5	15	5	5
1b	10	12	3	8	7	10
1c	7	11	7	16	6	3
2a	15	8	2	14	10	1
2b	8	8	9	13	10	2
3	16	3	6	19	3	3

Table 5: Performance of DisCover relative to CAARD and DSP for ambiguous queries. The entries are the number of instances.

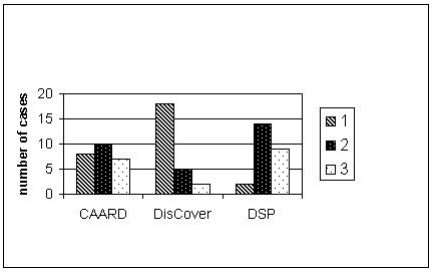
Question number	relative to CAARD			relative to DSP		
	better	equal	worse	better	equal	worse
1a	10	10	5	20	4	1
1b	11	12	2	18	6	1
1c	4	13	8	12	8	5
2a	9	8	8	18	7	0
2b	6	6	13	15	6	4
3	11	6	8	19	5	1

Table 6: Performance of DisCover relative to CAARD and DSP for popular queries. The entries are the number of instances.

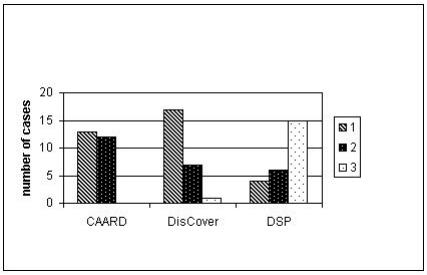
7. CONCLUSIONS

In this paper, we propose a new algorithm for hierarchical monothetic document clustering for summarization and browsing of Web search results. We select a monothetic clustering algorithm because they are well suited for generating concept hierarchies. With a focus on end-user requirements, we define a set of desirable properties for such an algorithm. We quantify some of the properties and frame an optimality criterion. Our algorithm differs from existing monothetic clustering algorithms in the optimality criterion. We empirically evaluate the performance of our algorithm in relation to two other monothetic clustering algorithms - CAARD and DSP based on a set of 10 queries. We also perform user studies to evaluate the performance of the algorithms on the more subjective criteria as well as to justify the selection of these evaluation criteria. Our empirical evaluations reveal that the nodes generated by DisCover cover documents more efficiently than either CAARD or DSP. Consequently, the reach time for DisCover tends to be significantly lower than that for CAARD and DSP. User studies also reveal that DisCover is more useful as a browsing and summarizing tool than either CAARD or DSP.

We use only coverage, distiveness and compactness properties



Popular Queries



Ambiguous Queries

Figure 4: Rank distribution of 3 algorithms for question 5.

to design the proposed algorithm. It will be interesting to see the effect of explicitly optimizing the reach time property on the algorithm. We have used a general purpose chunking tool for feature extraction in our experiments because the searches we performed and the corpus are general in nature. One may need to use domain dependent annotators or chunkers in order to apply this algorithm when used in generating concept hierarchies from search results obtained using domain-dependent queries on domain-specific corpora. It will be interesting to explore how to use domain-specific ontologies, if available, in generating the concept hierarchies.

8. REFERENCES

[1] G. Ball and D. A. Hall. A clustering technique for summarizing multivariate data. *Behavioral Science*, 12:153–155, 1967.

[2] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley and Sons, Inc., New York, 2001.

[3] K. Franzen and J. Karlgren. Verbosity and interface design. *Technical Report T2000:04, Swedish Institute of Computer Science (SICS)*, 2000.

[4] H. Frigui and O. Nasraoui. Simultaneous categorization of text documents and identification of cluster-dependent keywords. In *Proceedings of FUZZIEEE*, pages 158–163, Honolulu, Hawaii, 2002.

[5] G. Grefenstette. *Explorations in Automatic Thesaurus Discovery*. Kluwer Academic Publishers, 1994.

[6] A. Griffiths, H. Luckhurst, and P. Willett. Using inter-document similarity information in document retrieval systems. *Journal of the American Society for Information Sciences*, 37:3–11, 1986.

[7] M. A. Hearst. Automated discovery of WordNet relations. In C. Fellbaum, editor, *WordNet: an Electronic Lexical Database*. MIT Press, 1998.

[8] M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *Proceedings of SIGIR*, pages 76–84, Zürich, CH, 1996.

[9] R. Krishnapuram and K. Kummamuru. Automatic taxonomy generation: Issues and possibilities. In *LNCS: Proceedings of Fuzzy Sets and Systems (IFSA)*, volume 2715, pages 52–63. Springer-Verlag Heidelberg, Jan. 2003.

[10] K. Kummamuru, A. K. Dhawale, and R. Krishnapuram. Fuzzy co-clustering of documents and keywords. In *Proceedings of FUZZIEEE*, St. Louis, MO, 2003.

[11] K. Kummamuru and R. Krishnapuram. A clustering algorithm for asymmetrically related data with its applications to text mining. In *Proceedings of CIKM*, pages 571–573, Atlanta, USA, 2001.

[12] D. J. Lawnie and W. B. Croft. Generating hierarchical summaries for web searches. citeseer.nj.nec.com/lawrie03generating.html.

[13] D. Lawrie, W. B. Croft, and A. Rosenberg. Finding topic words for hierarchical summarization. In *Proceedings of SIGIR*, pages 349–357. ACM Press, 2001.

[14] D. J. Lawrie and W. B. Croft. Generating hierarchical summaries for web searches. In *Proceedings of SIGIR*, pages 457 – 458, 2003.

[15] B. Mandhani, S. Joshi, and K. Kummamuru. A matrix density based algorithm to hierarchically co-cluster documents and words. In *Proceedings of WWW* , Budapest, Hungary, 2003.

[16] F. C. N. Pereira, N. Tishby, and L. Lee. Distributional clustering of English words. In *Meeting of the Association for Computational Linguistics*, pages 183–190, 1993.

[17] M. F. Porter. An algorithm for suffix stripping. *Program*, 14:130–137, 1980.

[18] G. Salton. *The SMART Retrieval Systems*. Prentice Hall, Englewood Cliffs, N.J., 1971.

[19] M. Sanderson. Word sense disambiguation and information retrieval. In *Proceedings of SIGIR*, pages 142–151, 1994.

[20] M. Sanderson and W.B.Croft. Deriving concept hierarchies from text. In *Proceedings of SIGIR*, pages 206–213, 1999.

[21] E. Selberg and O. Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of WWW*, Darmstadt, Germany, December 1995.

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

[23] S. Vaithyanathan and B. Dom. Model selection in unsupervised learning with applications to document clustering. In *The Sixth International Conference on Machine Learning (ICML- 1999)*, pages 423–433, June 1999.

[24] S. Vaithyanathan and B. Dom. Model-based hierarchical clustering. In *Proceedings of Sixth Conference on Uncertainty in Artificial Intelligence*, pages 599–608, 2000.

[25] R. Weischedel, M. Meteer, R. Schwartz, L. Ramshaw, and J. Palmucci. Coping with ambiguity and unknown words through probabilistic models. *Association for Computational Linguistics*, 19(2):359–382, 1993.

[26] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of SIGIR*, pages 46–54, 1998.

[27] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of CIKM*, pages 515–524. ACM Press, 2002.