`Submitted to` *KDD-97*

Fast and Intuitive Clustering of Web Documents

Oren Zamir Oren Etzioni Omid Madani Richard M. Karp

Department of Computer Science & Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350
zamir@cs.washington.edu

Abstract

Conventional document retrieval systems (*e.g.*, Alta Vista) return long lists of ranked documents in response to user queries. Recently, document clustering has been put forth as an alternative method of organizing the results of a retrieval [4]. A person browsing the clusters can discover patterns that would be overlooked in the traditional ranked-list presentation.

In this context, a document clustering algorithm has two key requirements. First, the algorithm ought to produce clusters that are *easy-to-browse* – a user needs to determine at a glance whether the contents of a cluster are of interest. Second, the algorithm has to be *fast* even when applied to thousands of documents with no preprocessing.

This paper describes several novel clustering methods, which intersect the documents in a cluster to determine the set of words (or phrases) shared by all the documents in the cluster. We report on experiments that evaluate these *intersection-based* clustering methods on collections of snippets returned from Web search engines. First, we show that *word-intersection clustering* produces superior clusters and does so faster than standard techniques. Second, we show that our $O(n \log n)$ expected time *phrase-intersection clustering* method produces comparable clusters and does so more than two orders of magnitude faster than word-intersection.

Key Words: Document Clustering, Intersection-based Clustering, Suffix Trees, World Wide Web, Scatter/Gather.

1 Introduction

Conventional document retrieval systems return long lists of ranked documents that users are forced to sift through to find documents relevant to their queries. On the Web, this problem is exacerbated by the high recall and low precision of search engines such as Alta Vista [5], Hotbot[14], *etc.* Moreover, the typical user has trouble formulating highly specific queries and does not take advantage of advanced search options. Finally, this problem gets worse as the Web continues to grow.

Instead of attempting to reduce the number of documents returned (*e.g.*, by filtering methods [25] or by advanced pruning options [24]) we attempt to make search engine results easy to browse. Following the approach outlined in [4], we investigate document clustering as a method that enables users to efficiently navigate through a large collection of search engine results. In addition, clustering enables the user to discover patterns and structure in the document set that could be overlooked in the traditional ranked-list presentation. We believe that a document clustering method in this context requires:

1. **Ease-of-browsing**: A user needs to determine at a glance whether the contents of a cluster are of interest.
2. **Speed**: Web "surfers" are impatient and expect results within seconds.
3. **Scalability**: The method should be able to quickly cluster thousands of documents.
4. **No Preprocessing**: Since clustering is applied to a dynamically generated set of documents, it cannot rely on preprocessing of the documents to improve its efficiency.
5. **Snippet-Capable**: The method should produce "reasonable" clusters even when it only has access to the short document snippets returned by the search engines. Most users are unwilling to wait for the system to download the original documents.

In this paper we describe and experimentally evaluate several novel clustering methods that meet the above requirements to varying degrees.

The rest of this paper is organized as follows: The next section is a short overview of document clustering. Sections 3 and 4 describe word-intersection clustering and phrase-intersection clustering respectively. Section 5 describes preliminary experiments that compare our novel methods to standard clustering algorithms on collections of snippets returned by Web search engines. We conclude with a discussion of related and future work.

2 Document Clustering

Document clustering has been traditionally investigated mainly as a means of improving document search and retrieval. Recently, a technique named Scatter/Gather [4, 12] introduced document clustering as a document browsing method. Our work follows the same paradigm.

Document clustering algorithms fall into one of two classes: hierarchical and non-hierarchical (*i.e.*, "flat") methods [20]. Hierarchical clustering methods [32] create a binary-tree like organization (a dendrogram) and have the advantage of allowing the clusters to be viewed at different levels of resolution. Hierarchical agglomerative clustering (HAC) methods start with each document in a cluster of its own, iterate by merging the two closest clusters at each step, and terminate when some halting criterion is reached. Typically, these are greedy algorithms with no backtracking. HAC methods differ in three principal respects: (1) the distance function between documents, (2) the distance function between clusters, and (3) the halting criterion.

HAC clustering methods require the definition of a similarity or distance function between two documents. Each document is typically represented as a weighted attribute vector, with each word in the entire document collection being an attribute in this vector. These long, sparse vectors represent the fact that we are operating in a high-dimensional space where we do not know in advance which attributes would be significant to our task. The *Dice*, *Jaccard*, and *Cosine* similarity measures all look at the dot product between two attribute vectors, and differ by their normalization methods [23].

Given a definition for the distance between two documents, there are many ways to define the distance between two clusters (*i.e.*, sets of documents). The characteristics of the resulting clustering[1] and the time complexity of the HAC algorithm are both greatly determined by the definition of this cluster distance [30]. The commonly used distance functions between clusters are *Single-Link* (which defines the distance as the *minimum* document distance between the two clusters), *Complete-Link* (the *maximum* distance) and *Group-Average* (the *average* distance).

Several halting criteria for HAC methods have been suggested [19], but typically they fail to produce the desired result of having a balanced trade-off between the number of clusters, their size and their cohesion.

HAC algorithms are typically slow when applied to large document collections. Single-Link [26, 22, 6] and Group-Average [30] methods typically take $O(n^2)$ time[2], while Complete-Link methods typically take $O(n^3)$ time [7]. In terms of quality, on the other hand, Complete-Link algorithms have been shown to perform well in comparative studies of document retrieval [29], as they tend to produce tightly bound clusters, *i.e.*, clusters in which all the documents strongly relate to one another. Single-Link, and to a lesser degree Group-Average methods, exhibit a tendency toward creating elongated clusters. Elongated clusters have the undesirable property that two documents can be in the same cluster even though the similarity between them is small. From our experience in the Web domain, algorithms that produce elongated clusters often result in one or two large clusters, plus many extremely small ones. The Group-Average methods have another potentially undesirable effect: clusters tend to have long attribute vectors (*i.e.*, contain many nonzero entries) and similarities can arise from many low-weighted terms. This can lead to unintuitive clusters. Nonetheless, Group-Average algorithms have often been the method of choice for document clustering, because their trade-off between speed and clustering quality appears to be the most satisfying.

The above discussion suggests that traditional document clustering methods fail to meet the requirements listed in the introduction. Often, the methods generate elongated clusters that are *not* easy to browse – it's difficult to determine at a glance what the contents of a given cluster are likely to be. Furthermore, $O(n^2)$ time clustering is likely to be too slow for

[1]A clustering is the set of clusters produced by a clustering algorithm.

[2]Throughout this paper n denotes the number of documents to be clustered. The number of words per document is assumed to be bounded by a constant.

Web users when $n = 1,000$ or more. Finally, our experience shows that standard techniques perform poorly on the short and "noisy" snippets of Web documents.

3 Word-Intersection Clustering

Word-intersection clustering (Word-IC) is a new method designed to address some of the problems mentioned above. In particular, Word-IC characterizes clusters by the set of words shared by *every* document in the cluster. As a result, the *centroid* of the cluster – the words common to all the documents in it – can be presented to the user as a meaningful description of the contents of the cluster. For example, for a query on the word "Clinton" a sample of the resulting cluster centroids include: {Clinton, budget, economy}, {Buchanan, commentary, Pat}, and {campaign, Bob, Dole}.

In more technical terms, Word-IC results in non-elongated clusters, has a well motivated halting criterion and captures a desirable trade-off between the number of clusters, their size and their cohesion. We now describe Word-IC in detail.

Word-IC is a HAC algorithm that relies on a novel *Global Quality Function* (GQF) to quantify the quality of a clustering. We use the GQF as a heuristic to guide the HAC algorithm and as a halting criterion. At each iteration of the HAC algorithm, the two clusters whose merge results in the highest increase in the GQF are merged. The algorithm terminates when no merge increases the GQF.

The definition of a cluster's cohesion is central to Word-IC. We define the *cohesion* of a cluster c, denoted by $h(c)$, as the number of words common to all the documents in the cluster. Notice that this measure is a characteristic of the cluster as a set, and not a function of pair-wise similarities.

We define the *score* $s(c)$ of a single cluster c to be the product of its size (the number of documents in the cluster) and its dampened cohesion. We dampen the cohesion to express its decreasing marginal significance and to normalize it. Empirically, we found the best results when using a sigmoid function that dampens the cohesion measure to a value between 0 and 1. The parameter β determines the slope of the sigmoid function. Its value can be viewed as determining the point where the score stops being affected by the cohesion measure, and remains affected only by the size of the cluster. Thus, β specifies a trade-off between the two factors determining a cluster's quality: its size and its cohesion. We define the score of a cluster of size 1 (a cluster composed of a single document) to be 0.

$$s(c) = |c| \times \frac{1 - e^{-\beta h(c)}}{1 + e^{-\beta h(c)}} \tag{1}$$

For a clustering C, the $GQF(C)$ is a product of three components: (a) $f(C)$ - A function that is proportional to the fraction of clustered documents in the collection (an unclustered document is defined as one in a cluster by itself). This component captures the notion that unclustered documents are "bad". (b) $1/g(|C|)$ - Where $g(|C|)$ is an increasing function in the number of clusters. This component captures the notion that the fewer clusters there are, the better. (c) $\sum_{c \in C} s(c)$ - The sum of the scores of all clusters in the clustering. Thus:

$$GQF(C) = \frac{f(C)}{g(|C|)} \sum_{c \in C} s(c) \tag{2}$$

Notice that the factors $1/g(|C|)$ and $\sum_{c \in C} s(c)$ in (2) create a tension between two extremes: having a small number of large clusters of low cohesion *vs.* many small clusters of high cohesion. The *GQF* thus provides a trade-off between these two extremes. We have investigated different functional forms for the components of the *GQF*; our experiments have revealed that good results are obtained if $f(C)$ is the ratio of the number of clustered documents in the collection to the overall number of documents in the collection, and $g(|C|)$ is the number of clusters of size two or more raised to the power of 0.5.

Word-IC can be performed in $O(n^2)$ time. The result is a *monothetic* classification: all the documents in a given cluster must contain certain terms if they are to belong to it [21]. These clusters are not elongated, as all the documents in a cluster are similar to all other members by at least the cohesion of the cluster.

Experimental results in section 5 show that Word-IC is faster and results in higher quality clusterings than the commonly used cosine-based Group-Average HAC algorithm.

4 Phrase-Intersection Clustering

Word-IC, following the standard document clustering approach, treats a document as a set of words. This approach disregards some of the information present in a document, for example the occurrence of phrases (*i.e.*, consecutive sequences of words).

Phrase-intersection clustering (Phrase-IC) is another Intersection-based approach that looks at the phrases that are common to a group of documents, as an indication of the group's cohesion. This approach treats a document as a sequence of words, with the premise that phrases found in the document can be useful both for the clustering algorithm and as an indication of the cluster's content. For example, for a query on the word "Clinton" a sample of the phrases found common to many documents includes: "progress on the aids pandemic", "democratic party", and "Hillary Rodham Clinton".

Another advantage of Phrase-IC is that there exist efficient algorithms (for example [13]) for discovering long substrings common to many documents. Thus, for certain definitions of cluster cohesion based on common phrases we can leverage off this potential computational advantage.

4.1 Phrase-Intersection Clustering using *GQF*

A small variation in the definition of the *GQF* allows us to perform Phrase-IC. We can change the definition of a cluster's *cohesion* to be the length of the longest phrase common to all the documents of the cluster, instead of the number of words they all have in common. Phrase-IC using *GQF* is a HAC method that uses this modified version of the *GQF* as the heuristic guiding it and as its halting criteria. Experimental results in section 5 show that this algorithm performs slightly better than Word-IC.

4.2 Phrase-Intersection Clustering using Suffix Trees

The HAC algorithms suggested so far have $O(n^2)$ time complexity, an obstacle to our speed and scalability goals. Phrase-IC using *suffix trees* is an $O(n \log n)$ expected time algorithm that results in a large speedup without much degradation in quality.

The suffix tree [31, 11] of a set of strings is a compact *trie* [10] containing all the suffixes of all the strings. In our application, we build a suffix tree of all the documents (each treated as a string). As we are interested in whole words, our data structure is a suffix tree on words rather than characters, thus saving both time and space. Edges of the suffix tree are labeled with non-empty strings. Edges from the same node are labeled with strings that start with different words. The label of a node is the string formed by the concatenation of edge labels on the path from the root to the node. Each suffix of a document corresponds to exactly one path from the root to the leaf whose label is that suffix. An internal node, therefore, represents the point where different suffixes diverge. Equal suffixes from different documents correspond to the same leaf node, and equal phrases from different documents correspond to the same node.

The space requirement of the suffix tree is $O(n)$. As the branching factor of a document suffix tree varies considerably between the different levels, we use different data structures at the different levels of the tree.

Each node of a suffix tree represents a group of documents and a phrase that is common to all of them; the sequence of words from the root to the node represents the common phrase, and all the documents that have a suffix "passing" through the node make up the group. Therefore, each node of the suffix tree can be viewed as a potential cluster. Each node is assigned a score that is a function of the length of the phrase, the words appearing in the phrase, and the number of documents in that cluster.

To construct the suffix tree, when a document is received all its suffixes are inserted into the tree one by one. The running time of this construction is at worst quadratic in n, but the observed time complexity of the construction is actually $O(n)$ because of the small expected constant work of inserting each suffix. This is due to the fact that the average depth of the tree is less than three even for trees of very large collections, and the character comparison process for inserting a suffix is expected to look only at a small number of words. These observations were validated empirically. While there are $O(n)$ time suffix tree construction algorithms in the literature [28, 11], the speedups, if any, obtained from these algorithms are not worth their memory and time overhead.

This construction method has another advantage: because we visit each node in the path defining a suffix as we insert it, we can update the scores of all the nodes, and keep an updated priority queue of them, as we build the tree. This takes full advantage of "free" CPU time that is available due to the delay between the arrival of the documents from the Web.

In short, the expected construction time of the suffix tree itself is $O(n)$, but the process of score updates takes $O(n \log n)$ expected time. This is due to the fact that each time we pass a node while inserting a suffix, we update its score and then (re)insert it into its proper position in a priority queue of all scored nodes. The number of score updates is expected to be $O(n)$ and the node (re)positioning in the priority queue takes $O(\log n)$ time.

We determine clusters directly from the suffix tree, by presenting selected clusters (nodes) ordered by their score. These selected clusters may overlap. We argue that this feature of our algorithm is actually advantageous to the user, as topics do overlap (*e.g.*, topics in book indices often overlap, not as the result of the editor's incompetence, but because several topics may be relevant to the same section). When selecting which clusters to display, we make sure the overlap between the selected clusters is not high. This process takes $O(n \log n)$, but the reporting of the top ranked clusters starts immediately. The suffix tree clustering algorithm, unlike the HAC methods, is not greedy in the sense that all possible clusters with a non-empty common phrase are identified and scored. However, the algorithm we use to select which clusters to display to the user is greedy.

5 Preliminary Experiments

In this section we compare the running time and the clustering quality of the different algorithms.

It is hard to calculate the quality of a clustering algorithm, as one has to know the "correct" clustering of the different test cases, and this is hard even with an external source of knowledge (such as human indexing). We chose to apply the algorithms to snippet collections created by merging several distinct base collections. We then scored the resulting clusterings by comparing them to the original partition of the snippets into base collections.

We created 88 base collections from the snippets returned by MetaCrawler [24] in response to 88 different queries. MetaCrawler is a parallel search engine – it routes queries to various different search engines and collates the results - thus assuring us of a wide and heterogeneous sample of Web documents. Each of the queries contained between 1 and 4 keywords and was chosen to define a certain topic in computer science (such as kernel & architecture; biology & computational; compiler). Each base collection contained approximately 100 snippets; each snippet contained 40 words, on average. Test collections were created by merging 1 to 8 randomly chosen base collections, giving us test collections ranging from 100 to 800 snippets in size. 20 test collections of each size were created, for a total of 200 test collections.

We need a scoring method to compare the original partition of the snippets into base collections with the algorithm generated clusterings. To do so, we look at all pairs of documents in a single cluster, and count the number of true-positive pairs (the two documents were also in the same base collection) and false-positive pairs (they were not in the same base collection).

Let B be a test collection, C be a clustering of the snippets in B, $u(C)$ be the number of unclustered documents in C, and $t(c)$ and $f(c)$ be the number of true-positive pairs and the number of false-positive pairs in cluster $c \in C$, respectively. The quality of the clustering C, denoted by $Q(C)$, is:

$$Q(C) = \frac{\sum_{c \in C}(\sqrt{t(c)} - \sqrt{f(c)}) - u(C)}{\sum_{c \in B}\sqrt{\binom{|c|}{2}}} \tag{3}$$

We use the square roots of $t(c)$ and $f(c)$ to avoid over-emphasizing larger clusters, as the number of document pairs in a cluster c is $\Theta(|c|^2)$. We subtract the number of unclustered documents, as these documents are misplaced. Finally, we normalize this score by the maximal score possible, which is simply the sum of the square roots of the number of pairs in the original base collections. The result is a score between -1 and 1 (with a negative number meaning more documents were misplaced than were put in their correct document group).

Note that this scoring metric works best when the clusters do not overlap. Therefore, when computing this metric for the suffix tree clustering method, we remove a document from all but the first cluster that it appears in (this puts suffix tree clustering on equal footing with the other methods by having it create a partition).

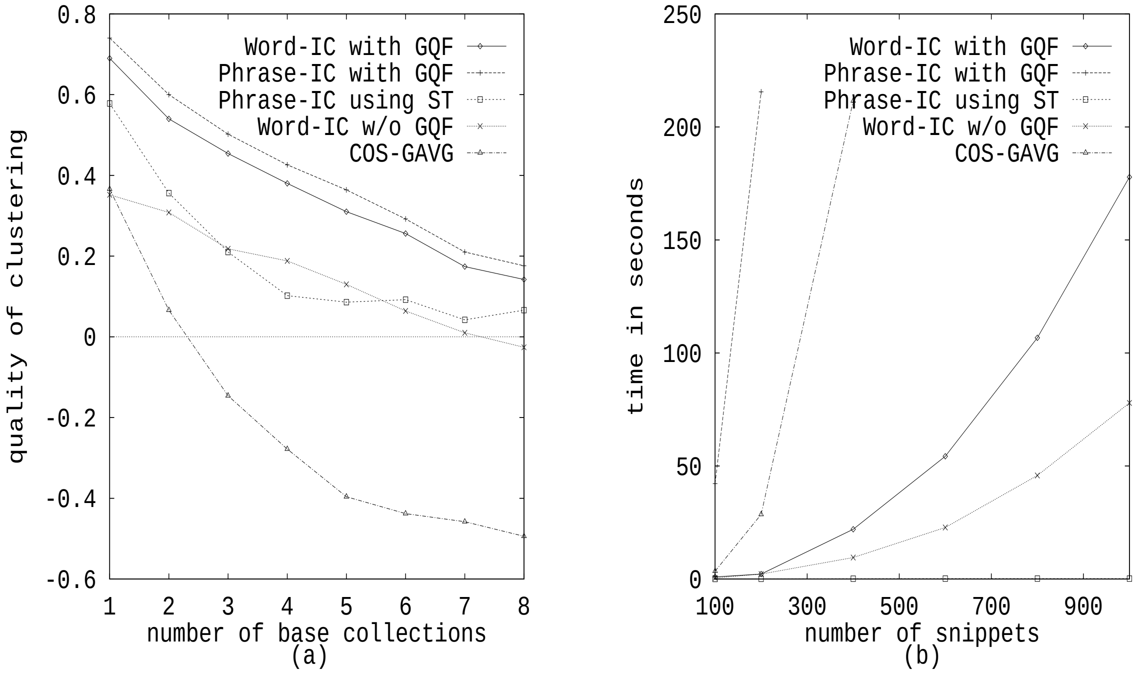

Figure 1: (a) The quality of the clusterings produced by the different algorithms. (b) The execution times of the different algorithms.

Figures 1(a) compares the quality of the clusters produced by the algorithms as a function of the number of base collections merged. We compare our clustering algorithms with the cosine-based Group-Average HAC algorithm (hereafter referred to as COS-GAVG), as it is one of the most commonly used algorithm for document clustering.

Word-IC, as defined in section 3, includes two principal components: the definition of cohesion and the GQF. We wish to investigate how the definition of cohesion alone influences the clustering. To do so, we compare the performances of a variation of the Word-IC algorithm that does not use the GQF. This algorithm defines the *similarity* of two clusters as the cohesion of the cluster that would be formed upon merging the two clusters, where cohesion is defined as in Word-IC. It then performs a HAC algorithm, merging at each step the two most similar clusters. It terminates with a halting criterion similar to the one used in the COS-GAVG algorithm.

To optimize the halting criterion of COS-GAVG and Word-IC without GQF, we ran preliminary experiments and chose the criterion that performed best.

All the algorithms show a quality degradation as the number of merged base collection increases. This is not surprising, as more merged base collections increase the difficulty of the clustering task. The COS-GAVG algorithm performed poorly in our experiments. The fact that we are using short, "noisy" snippets, might contribute to the poor quality of its results. Both Word-IC and Phrase-IC using the GQF show the highest quality results. The advantages of the *GQF* can be seen by comparing Word-IC without *GQF* with the regular Word-IC. The suffix tree clustering algorithm is shown to produces results that are not much worse than the intersection-based ones.

To compare the speed of the algorithms, we clustered snippet collections of 100 to 1000 snippets using a DEC Alpha-station 500, 333 MHz, with 320M RAM. The algorithms were implemented in C++ and were optimized to the same degree (with the exception of Phrase-IC – see below).

Figure 1(b) presents the results of this experiment. The times measured are the actual times spent clustering, without including periods when the system was idle, waiting for documents to arrive from the Web. The COS-GAVG algorithm is slower than the intersection-based ones as it requires long attribute vectors. Using *GQF* adds a constant factor to the execution time of Word-IC because of the added complexity. Phrase-IC using *GQF* has not been optimized but is expected to be slower than the COS-GAVG method even after optimization. The performance of the suffix tree clustering algorithm cannot be seen on the scale shown, as it clusters 1000 snippets in less than 0.25 seconds. It should be mentioned that further optimizations could be carried out and might affect the different algorithms to different degrees, but we expect the relations between them to remain largely unchanged.

6 Future Work

Below, we consider several observations about our clustering methods and the directions they suggest for future work.

Suffix tree clustering has a great advantage when clustering Web documents because the suffix tree can be built incrementally as the documents arrive. This allows the use of "free" CPU cycles as the system waits for the documents. We have adapted a suffix tree construction algorithm that allows us to also score and sort the potential clusters incrementally, as the documents arrive. This construction, though having poor worst-case behavior, can be shown to have $O(n \log n)$ expected time behavior in our domain. We plan to experimentally compare the speed of different suffix tree algorithms in future work.

While experimenting with the system we have found that certain queries lend themselves very nicely to Phrase-IC, while other queries do not. We also found that Word-IC and Phrase-IC often yield complementary presentations of the collection to the user and need not be viewed as alternatives. For small collections, we could allow the user to view the results of both algorithms. Will users find multiple distinct clusterings worthwhile? Finally, a question that still has to be answered is how does clustering whole documents compare with clustering snippets. Will this result in a substantial improvement in cluster quality? Will such an improvement outweigh the increased delay? Ultimately, the answers to these questions are subjective. We plan to deploy a clustering module on top of MetaCrawler, which will enable us to conduct user studies aimed at answering these questions empirically.

7 Related Work

Dynamic document clustering of online search outputs was introduced in [4], and investigated in [16] as well, as a means to identify near-duplicate documents. [12] describes user studies on the Scatter/Gather system that demonstrate the systems usefulness.

The Scatter/Gather system uses a linear time partition algorithm that relies on clusters created by applying the COS-GAVG algorithm to a sample (of size $\sqrt{n}$) of the collection. Therefore, the quality of its results are likely to be lower than that of a full COS-GAVG algorithm. We have demonstrated that our intersection-based approach produces results of higher quality in some cases.

Conceptual clustering is a clustering method meant to produce clusters that are easy to understand by restricting the clustering algorithm to only consider clusters that can be characterized through logical combinations of predicates in a particular predicate language [27]. Word-IC and Phrase-IC can be viewed as following a similar approach in that the clusters created can be defined in a clear and simple manner, but they are faster and do not require the structured background knowledge needed for conceptual clustering.

Measuring the global quality of a partition of the data in a Bayesian context appeared in COBWEB [9], in AUTOCLASS [3], as well as in other systems [1]. These clustering techniques are typically slow and do not perform well in domains with sparse attribute vectors. Only recently has an attempt been made to apply these Bayesian quality measures to guide a HAC algorithm in the document clustering domain [15], but more comprehensive comparative studies are required in order to determine the benefit of this method.

Various other methods have been investigated in the document clustering domain. The non-hierarchical ones include single-pass methods such as *assign-to-nearest*, as well as multi-pass methods such as *iterative-seed-selection* that is used in CLUSTER [18] and in the *Cover-Coefficient* method [2]. These methods are fast, but have shown poor results for document clustering.

MetaCrawler and Excite [8] allow the user to view the documents returned from queries either as a ranked list or sorted according to site. This can be viewed as a very basic form of clustering. Alta Vista [5] has introduced LiveTopics, a query refinement tool that performs word frequency analysis. It does not perform document clustering but addresses the problem of helping the user handle the large number of documents typically returned by the search engine in a different way.

8 Conclusion

We have attempted to address the hard problem of enabling users to quickly navigate through the results of Web search engines. We have described and experimentally evaluated two novel clustering methods: word- and phrase- intersection clustering. Phrase-intersection clustering using suffix trees is an $O(n \log n)$ expected time algorithm that appears promising in terms of the stringent requirements outlined in the introduction including ease of browsing, speed, and scalability. In addition, we believe that our clustering methods will enable users to discover patterns and find structure in search engine responses that are not readily apparent in the standard ranked-list presentation. Of course, additional experiments and extensive user studies are necessary before we can make definitive claims about the performance of our algorithms in practice.

Acknowledgments

We thank Ronen Feldman for early discussions related to knowledge discovery in texts. We would like to thank Neal Lesh, Jonathan Shakes, Adam Carlson and Sujay Parekh for their comments on earlier drafts. Erik Selberg deserves many thanks for his help in integrating this system with the MetaCrawler. Thanks are due to all members of the UW AI group for their helpful comments and ideas throughout this research. Zhenya Sigal made important contributions to the implementation and its users interface. This research was funded in part by Office of Naval Research grant 92-J-1946, by ARPA / Rome Labs grant F30602-95-1-0024, by a gift from Rockwell International Palo Alto Research, and by National Science Foundation grant IRI-9357772.

References

[1] J. R. Anderson and M. Matessa. An iterative bayesian algorithm for categorization. In D. Fisher and M. Pazzani, editors, *Concept formation: Knowledge and experience in unsupervised learning*. Morgan Kaufman, 1991.

[2] F. Can. Concept and effectiveness of the cover-coefficient-based clustering methodology for text databases. *ACM Trans. On database systems*, 15:483–517, 1990.

[3] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: A bayesian classification system. In *Proceedings of the Fifth International Machine Learning Conference*, pages 54–64, 1988.

[4] D. R. Cutting, D. R. Karger, J. O. Pederson, and J. W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *15th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–29, 1992.

[5] Digital Equipment Corporation. Altavista home page. See `http://www.altavista.com`.

[6] E. W. Dijkstra. The problem of the shortest subspanning tree. In *A Discipline of Programming*, pages 154–60. Prentice Hall, 1976.

[7] A. El-Hamdouchi and P. Willet. Techniques for the measurement of clustering tendency in document retrieval systems. *J. Information Science*, 13:361–65, 1989.

[8] Excite, Inc. Excite home page. See `http://www.excite.com`.

[9] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–192, 1987.

[10] E. Fredkin. Trie memory. *Communications of the ACM*, 3:490–499, 1960.

[11] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, chapter 6. Cambridge University Press, 1997.

[12] M. A. Hearst and J. O. Pederson. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *19th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 76–84, 1996.

[13] L. C. K. Hui. Color set size problem with applications to string matching. In *Combinatorial Pattern Matching Third Annual Symposium*, 1992.

[14] Inktomi, Inc. Hotbot home page. See `http://www.hotbot.com`.

[15] M. Iwayama and T. Tokunaga. Cluster-based text categorization: a comparison of category search techniques. In *18th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1995.

[16] J.W. Kirriemuir and P. Willet. Identification of duplicate and near-duplicate full-text records in database search-outputs using hierarchic cluster analysis. In *18th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 241–256, 1995.

[17] Michael Mauldin. Lycos home page. See `http://lycos.cs.cmu.edu`.

[18] R. S. Michalski and R. E. Stepp. Automated construction of classifications: conceptual clustering versus numerical taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, PAMI-5, no.4:396–410, 1983.

[19] G. W. Milligan and M. C. Cooper. An examination of procedures for detecting the number of clusters in a data set. *Psychometrika*, 50:159–79, 1985.

[20] E. Rasmussen. Clustering algorithms. In W.B. Frakes and R. Baeza-Yates, editors, *Information Retrieval*, pages 419–442. Prentice Hall, Eaglewood Cliffs, N.J., 1992.

[21] C.J. Van Rijsbergen. *Information Retrieval.* London, Butterworths, 1979.

[22] C.J. Van Rijsbergen. An algorithm for information structuring and retrieval. *Computer Journal*, 14:407–412, 71.

[23] G. Salton. *Automatic text processing.* Addison-Wesley, 1989.

[24] Erik Selberg and Oren Etzioni. Multi-service search and comparison using the metacrawler. In *Proc. 4th World Wide Web Conf.*, pages 195–208, Boston, MA USA, 1995. See `http://www.cs.washington.edu/research/metacrawler`.

[25] J. Shakes, M. Langheinrich, and O. Etzioni. Ahoy! the home page finder. In *Proc. 6th World Wide Web Conf.*, Santa Clara, CA USA, 1997. See `http://www.cs.washington.edu/research/ahoy`.

[26] R. Sibson. Slink: an optimally efficient algorithm for the single link cluster method. *Computer Journal*, 16:30–34, 1973.

[27] R. E. Stepp and R. S. Michalski. Conceptual clustering. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning, volume II*, pages 371–392. Morgan Kaufmann, 1986.

[28] Esko Ukkonen. On-line construction of suffix-trees. *Algorithmica*, 14:249–260, 1995.

[29] E.M. Voorhees. *The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval*. PhD thesis, Cornell University, 1986.

[30] E.M. Voorhees. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing & Management*, 22:465–476, 1986.

[31] P. Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–11, 1973.

[32] P. Willet. Recent trends in hierarchical document clustering: a critical review. *Information Processing and Management*, 24:577–97, 1988.