From: KDD-97 Proceedings. Copyright © 1997, AAAI (www.aaai.org). All rights reserved.

Fast and Intuitive Clustering of Web Documents*

Oren Zamir and **Oren Etzioni** and **Omid Madani** and **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 retrieval results (Cutting *et al.* 1992). A person browsing the clusters can discover patterns that could be overlooked in the traditional presentation.

This paper describes two novel clustering methods that 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 $O(n \log n)$ time *phrase-intersection clustering* method produces comparable clusters and does so more than two orders of magnitude faster than all methods tested.

Introduction

Conventional document retrieval systems return long lists of ranked documents that users are forced to sift through to find the relevant documents. On the Web, this problem is exacerbated by the high recall and low precision of search engines (*e.g.*, Alta Vista). 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 (Shakes, Langheinrich, & Etzioni 1997)) we attempt to make search engine results easy to browse. We investigate document clustering as a method that enables users to efficiently navigate through a large collection of documents. In addition, clustering enables the user to discover patterns and structure in the collection that could be overlooked in the traditional ranked-list presentation. In this context, a document clustering method requires:

1. **Ease-of-browsing**: A user needs to determine at a glance whether a cluster's contents are of interest.
2. **Speed:** Web users expect results within seconds.
3. **Scalability:** The method should be able to quickly cluster thousands of documents.
4. **Snippet-tolerance**: 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 two novel clustering methods that meet the above requirements to varying degrees.

Document Clustering

Document clustering has been traditionally investigated mainly as a means of improving document search and retrieval. Recently, a technique named Scatter/Gather (Cutting *et al.* 1992) introduced document clustering as a document browsing method. Our work follows the same paradigm.

Hierarchical agglomerative clustering (HAC) algorithms are the most commonly used methods for document clustering (Willet 1988). These algorithms start with each document in a cluster of its own, iterate by merging the two most similar clusters, and terminate when some halting criterion is reached.

HAC algorithms require the definition of a similarity function between documents and between sets of 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. The similarity of two documents is often taken as a normalized function of the dot product of their attribute vectors.

Several halting criteria for HAC algorithms have been suggested (Milligan & Cooper 1985), but they are typically based on predetermined constants (*e.g.*, halt when 5 clusters remain). Because the HAC algorithm

Copyright 1997, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

does not backtrack it is very sensitive to the halting criterion — when the algorithm mistakenly merges two "good" clusters, the resulting cluster could be meaningless to the user. In the domain of search engines, we often receive many irrelevant snippets — snippets that do not have any correlation to the query or to other snippets. This sort of "noise" reduces even further the effectiveness of commonly-used halting criteria.

HAC algorithms are typically slow when applied to large document collections. Single-link (Rijsbergen 1971) and group-average methods typically take $O(n^2)$ time[1], while complete-link methods typically take $O(n^3)$ time (Voorhees 1986b). In terms of quality, on the other hand, complete-link algorithms have been shown to perform well in comparative studies of document retrieval (Voorhees 1986a), as they tend to produce "tight" clusters — clusters in which all the documents are similar 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. This can lead to non-intuitive clusters.

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 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.

Word-Intersection Clustering

Word-intersection clustering (Word-IC) is a new method designed to address some of the problems mentioned above. Word-IC results in "tight" clusters, has a well motivated halting criterion and captures a desirable trade-off between the number of clusters, their size and their cohesion.

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 the heuristic to guide the HAC algorithm and as the halting criterion. At each iteration of the HAC algorithm, the two clusters whose union would result in the highest increase in the *GQF* are merged. The algorithm terminates when no merge increases the *GQF*. Next we'll describe the *GQF*.

The definition of a cluster's cohesion is central to the *GQF*. We define the *cohesion* of a cluster c as the number of words common to all the documents in the cluster. We define the *score* $s(c)$ of a single cluster c to be the product of its size $|c|$ and its dampened cohesion. The score of a singleton cluster is defined to be 0.

For a clustering C, the $GQF(C)$ is a product of three components: (a) $f(C)$ — A function proportional to the fraction of documents in non-singleton clusters. This component captures the notion that singleton clusters are "bad". (b) $1/g(|C|)$ — Where $g(|C|)$ is an increasing function in the number of non-singleton 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:

$$GQF(C) = \frac{f(C)}{g(|C|)} \sum_{c \in C} s(c) \qquad (1)$$

Notice that the factors $1/g(|C|)$ and $\sum_{c \in C} s(c)$ create a tension between two extremes: having a small number of large clusters of low cohesion *vs.* having many small clusters of high cohesion. The *GQF* 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 simply the ratio of the number of documents in non-singleton clusters to the overall number of documents, and $g(|C|)$ is the number of non-singleton clusters 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. In Word-IC, that set of common words — the *centroid* of the cluster — can be presented to the user as a concise description of its contents. We believe that this approach results in high-quality clusters because all the documents in the cluster share at least the words in its centroid.

Experimental results in section 5 show that Word-IC is faster and results in higher quality clusters than the commonly used group-average HAC algorithm using the cosine inter-document similarity function.

Phrase-Intersection Clustering using Suffix Trees

Following the standard document clustering paradigm, Word-IC treats a document as a *set* of words, disreg-

[1] 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.

arding word *sequences*. We conjecture that word proximity information may be valuable in some cases. Furthermore, clusters whose centroid is a shared phrase would be particularly easy to browse. Based on these observations we formulate Phrase-intersection clustering (Phrase-IC) — a novel intersection-based approach that looks at the phrases that are common to a group of documents as an indication of the group's cohesion.

The HAC algorithms mentioned previously have $O(n^2)$ time complexity, an obstacle to our speed and scalability goals. Phrase-IC using *suffix trees* (Weiner 1973) is an $O(n \log n)$ time algorithm that results in a large speedup without much degradation in quality.

The suffix tree of a set of strings is a *compact trie* containing all the suffixes of all the strings. In our application, we construct a suffix tree of all the documents. Each node of the suffix tree represents a group of documents and a phrase that is common to all of them; the label of the node represents the common phrase, and all the documents who have corresponding leaves that are descendants of the node make up the group. Therefore, each node 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 it, and the number of documents in that cluster. The nodes are sorted based on their score.

Clusters are determined directly from this sorted list of potential clusters using a simple selection algorithm. Notice that the selected clusters may overlap. We believe that this feature is advantageous to the user, as many topics do overlap. When selecting which clusters to display, we make sure the overlap between the selected clusters is not high. We are currently exploring the option of merging potential clusters with high overlap.

The space requirement of the suffix tree is $O(n)$, and it can be constructed in $O(n)$ time (Ukkonen 1995). The suffix tree can be built incrementally as the documents arrive. This allows the use of "free" CPU cycles as the system waits for additional documents. The number of potential clusters is $O(n)$, thus sorting them and selecting which to present to the user can be performed in $O(n \log n)$ time.

Preliminary Experiments

It is hard to measure the quality of a clustering algorithm, as one has to know the "correct" clustering of the test cases. We chose to apply the algorithms to snippet collections created by merging several distinct base collections. We then scored the resulting clustering by measuring its deviation from the original partition of the snippets into base collections.

We created 88 base collections from snippets returned by MetaCrawler (Selberg & Etzioni 1995) in response to 88 different queries. Each of the queries contained between 1 and 4 keywords and defined a topic in computer science (*e.g.* kernel & architecture; biology & computational; compiler). Each base collection contained approximately 120 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 120 to 1000 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 clustering. 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. The quality of the clustering is a function of the difference between these two quantities, normalized by the size of the collection clustered. A quality score of 1 means a perfect reproduction of the original partition.

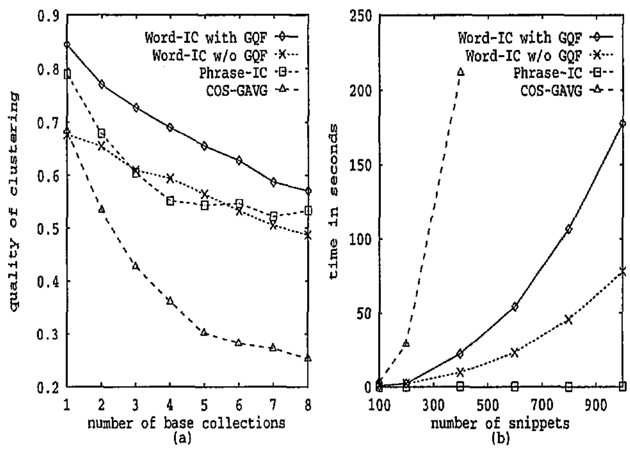

Figure 1: (a) The quality of the clusters produced by the different algorithms. (b) The execution time of the different algorithms. The execution time of the Phrase-IC algorithm cannot be seen on the scale shown, as it clusters 1000 snippets in less than 0.5 seconds. This algorithm exhibits a good tradeoff between quality and speed — it achieves high quality clusters in $O(n \log n)$ time.

Figure 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 group-average HAC algorithm using the cosine inter-document similarity function (referred to as COS-GAVG), which is one of the most commonly used document clustering algorithms.

Word-IC includes two principal components: the definition of cohesion and the *GQF*. To investigate how the definition of cohesion influences the clustering, we measured the performance 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 performs a HAC algorithm, merging at each step the two most similar clusters, and terminates with the same halting criterion used in COS-GAVG. At the top of Figure 1(a) we see that omitting GQF degrades the performance of Word-IC.

All the algorithms show, as expected, a quality degradation as the number of merged base collections increases. The COS-GAVG algorithm performs poorly in our experiments. The fact that we are clustering short, "noisy" snippets, probably contributes to the poor quality of its results. Word-IC shows the highest quality results. The advantages of the GQF can be seen by comparing Word-IC without GQF with the regular Word-IC. We believe this is mainly due to the well-motivated halting criterion of the algorithm. The suffix tree clustering algorithm produces results that are not much worse than those produced by Word-IC.

To compare the execution time 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.

Figure 1(b) presents the results of this experiment. The times measured are the actual times spent clustering, without including idle periods when the system was waiting for snippets to arrive. The COS-GAVG algorithm is slower than the intersection-based ones as it requires longer attribute vectors. Using GQF adds a constant factor to the execution time of Word-IC because of the added complexity. The performance of the suffix tree clustering algorithm cannot be seen on the scale shown, as it clusters 1000 snippets in less than 0.5 seconds.

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 and need not be viewed as alternatives; we could allow the user to view the results of both algorithms. An interesting question is whether users will find multiple sets of clusters worthwhile, and what visualization techniques would be best for this task.

Finally, a question that has to be answered is how would the clustering results change if we download the original documents from the Web. Will this result in a substantial improvement in quality, and will such an improvement outweigh the increased delay? We are currently deploying a clustering module on top of MetaCrawler, which will enable us to conduct user studies aimed at answering these questions empirically.

Conclusion

We have described and experimentally evaluated two novel clustering methods that enable users to quickly navigate through the results of Web search engines: word- and phrase- intersection clustering. Phrase-IC using suffix trees is an $O(n \log n)$ time algorithm that appears promising in terms of the stringent requirements outlined in the introduction including ease of browsing, speed, and scalability. Of course, additional experiments and extensive user studies are necessary before definitive claims can be made about the performance of our algorithms in practice.

Acknowledgments: We thank Ronen Feldman for early discussions and Erik Selberg for his help in integrating this system with MetaCrawler. Zhenya Sigal made important contributions to the implementation. 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

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

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

Rijsbergen, C. V. 1971. An algorithm for information structuring and retrieval. *Computer Journal* 14:407–412.

Selberg, E., and Etzioni, O. 1995. Multi-service search and comparison using the metacrawler. In *Proc. 4th World Wide Web Conf.*, 195–208.

Shakes, J.; Langheinrich, M.; and Etzioni, O. 1997. Ahoy! the home page finder. In *Proc. 6th World Wide Web Conf.*

Ukkonen, E. 1995. On-line construction of suffix-trees. *Algorithmica* 14:249–260.

Voorhees, E. 1986a. *The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval.* Ph.D. Dissertation, Cornell University.

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

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

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