International Journal of Research Studies in Science, Engineering and Technology
Volume 2, Issue 6, June 2015, PP 1-13
ISSN 2349-4751 (Print) & ISSN 2349-476X (Online)

A Described Feasibility Analysis on Web Document Clustering

Ganesh Kr Yadav
SITM, Lucknow
gky14jul@gmail.com

Amit Kumar
SRGI, Jhansi
amitkumar41@gmail.com

Abstract: *The paper articulates the unique requirements of Web document clustering and reports on the first evaluation of clustering methods in this domain. A key requirement is that the methods create their clusters based on the short snippets returned by Web search engines. Surprisingly, we find that clusters based on snippets are almost as good as clusters created using the full text of Web documents. To satisfy the stringent requirements of the Web domain, we introduce an incremental, linear time (in the document collection size) algorithm called Suffix Tree Clustering (STC), which creates clusters based on phrases shared between documents. We show that STC is faster than standard clustering methods in this domain, and argue that Web document clustering via STC is both feasible and potentially beneficial.*

1. INTRODUCTION

Conventional document retrieval systems return long lists of ranked documents that users are forced to sift through to find relevant documents. The majority of today's Web search engines (*e.g.*, Excite, AltaVista) follow this paradigm. Web search engines are also characterized by extremely low precision.

The low precision of the Web search engines coupled with the ranked list presentation make it hard for users to find the information they are looking for. Instead of attempting to increase precision (*e.g.*, by filtering methods - Shakes et. al., 97 - or by advanced pruning options - Selberg and Etzioni, 95) we attempt to make search engine results easy to browse. This paper considers whether document clustering is a feasible method of presenting the results of Web search engines.

Many document clustering algorithms rely on off-line clustering of the entire document collection (e.g., Cutting et. al., 93; Silverstein and Pedersen, 97), but the Web search engines' collections are too large and fluid to allow off-line clustering. Therefore clustering has to be applied to the much smaller set of documents returned in response to a query. Because the search engines service millions of queries per day, free of charge, the CPU cycles and memory dedicated to each individual query are severely curtailed. Thus, clustering has to be performed on a separate machine, which receives search engine results as input, creates clusters and presents them to the user.

Based on this model, we have identified several key requirements for Web document clustering methods:

- **Relevance:** The method ought to produce clusters that group documents relevant to the user's query separately from irrelevant ones.
- **Browsable Summaries:** The user needs to determine at a glance whether a cluster's contents are of interest. We do not want to replace sifting through ranked lists with sifting through clusters. Therefore the method has to provide concise and accurate descriptions of the clusters.
- **Overlap:** Since documents have multiple topics, it is important to avoid confining each document to only one cluster (Hearst, 98).
- **Snippet-tolerance:** The method ought to produce high quality clusters even when it only has access to the snippets returned by the search engines, as most users are unwilling to wait while the system downloads the original documents off the Web.
- **Speed**: A very patient user might sift through 100 documents in a ranked list presentation. We want clustering to allow the user to browse through at least an order of magnitude more documents. Therefore the clustering method ought to be able to cluster up to one thousand snippets in a few seconds. For the impatient user, each second counts.

- **Incrementality:** To save time, the method should start to process each snippet as soon as it is received over the Web.

Below, we introduce *Suffix Tree Clustering* (STC) - a novel, incremental, $O(n)$[1] time algorithm designed to meet these requirements. STC does not treat a document as a set of words but rather as a string, making use of proximity information between words. STC relies on a *suffix tree* to efficiently identify sets of documents that share common phrases and uses this information to create clusters and to succinctly summarize their contents for users.

To demonstrate the effectiveness and speed of STC, we have created MetaCrawler-STC, a prototype clustering Web search engine, which is accessible at the following URL: http:/ www.cs.washington. edu/research/clustering. MetaCrawler-STC takes the output of the MetaCrawler meta search engine (Selberg and Etzioni, 95) and clusters it using the STC algorithm. Figure 1 shows sample output for MetaCrawler-STC. We provide preliminary experimental evidence that STC satisfies the speed, snippet tolerance, and relevance requirements, and that it benefits from creating overlapping clusters. We believe the shared phrases of a cluster provide an informative way of summarizing its contents, but a user study to validate this belief is an area for future work.

search results for the query: "salsa" documents: 246, clusters: 15		
Cluster num.	Size	Shared phrases and sample document titles
1	8	**Puerto Rico; Latin Music** *1. Salsa Music in Austin* *2. LatinGate Home Page*
2	20	**Follow Ups post; York Salsa Dancers** *1. Origin and Development of Salsa?* *2. Re: New York Salsa Dancers are the best because...*
3	40	**music; entertainment; latin; artists** *1. Latin Midi Files Exchange* *2. Salsa Music on the Web. Con Sabor!*
4	79	**hot; food; chiles; sauces; condiments; companies** *1. Religious Experience Salsa* *2. Arizona Southwestern Cuisine and Gifts*
5	41	**pepper; onion; tomatoes** *1. Salsa Mixes* *2. Salsa Q & A*

Fig1. *The output of our MetaCrawler-STC clustering engine for the query "salsa". Only the first 5 clusters are shown here. The words in* ***bold*** *are the shared phrases found in the clusters. Note the descriptive power of phrases such as "Puerto Rico", "Latin Music" and "York Salsa Dancers".*

The paper is organized as follows: The next section is an assessment of the extent to which previous work on document clustering meets the requirements of the Web domain. The following section ie section-1 introduction of paper, section-2 existing work on document clustering, section-3 proposed work, section-4 experimental and execution and section-5 described conclusion in detail. Subsequently, we report on the experimental evaluation of STC and other clustering algorithms in the Web domain. We conclude by summarizing our contributions and with directions for future work.

2. EXISTING WORK ON DOCUMENT CLUSTERING

In this section we review previous work on document clustering algorithms and discuss how these algorithms measure up to the requirements of the Web domain.

Document clustering has been traditionally investigated mainly as a means of improving the performance of search engines by pre-clustering the entire corpus (the cluster hypothesis - van Rijsbergen, 79). However, clustering has also been investigated as a post-retrieval document browsing technique (Croft, 78; Cutting et. al, 92; Allen et. al., 93; Leouski and Croft, 96). Our work follows this alternative paradigm.

Numerous documents clustering algorithms appear in the literature (see Willet, 88 for review). *Agglomerative Hierarchical Clustering* (AHC) algorithms are probably the most commonly used.

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.

These algorithms are typically slow when applied to large document collections. *Single-link* and *group-average* methods typically take $O(n^2)$ time, while *complete-link* methods typically take $O(n^3)$ time (Voorhees, 86). As our experiments demonstrate, these algorithms are too slow to meet the speed requirement for one thousand documents.

Several halting criteria for AHC algorithms have been suggested (Milligan and Cooper, 85), but they are typically based on predetermined constants (*e.g.*, halt when 5 clusters remain). These algorithms are very sensitive to the halting criterion - when the algorithm mistakenly merges multiple "good" clusters, the resulting cluster could be meaningless to the user. In the Web domain, where the results of queries could be extremely varied (in the number, length, type and relevance of the documents), this sensitivity to the halting criterion often causes poor results. Another characteristic of the Web domain is that we often receive many outliers. This sort of "noise" reduces even further the effectiveness of commonly used halting criteria.

Linear time clustering algorithms are the best candidates to comply with the speed requirement of on-line clustering. These include the K-Means algorithm - $O(nkT)$ time complexity where k is the number of desired clusters and T is the number of iterations (Rocchio, 66), and the Single-Pass method - $O(nK)$ were K is the number of clusters created (Hill, 68). One advantage of the K-Means algorithm is that, unlike AHC algorithms, it can produce overlapping clusters. Its chief disadvantage is that it is known to be most effective when the desired clusters are approximately spherical with respect to the similarity measure used. There is no reason to believe that documents (under the standard representation as weighted word vectors and some form of normalized dot-product similarity measure) should fall into approximately spherical clusters. The Single-Pass method also suffers from this disadvantage, as well as from being order dependant and from having a tendency to produce large clusters (Rasmussen, 92). It is, however, the most popular incremental clustering algorithm (as can be seen from its popularity in the event detection domain - see TDT, 97).

Buckshot and Fractionation are fast, linear time clustering algorithms introduced in (Cutting et. al., 92). Fractionation is an approximation to AHC, where the search for the two closest clusters is not performed globally, but in rather locally and in a bound region. This algorithm will obviously suffer from the same disadvantages of AHC - namely the arbitrary halting criteria and the poor performance in domains with many outliers. Buckshot is a K-Means algorithm where the initial cluster centroids are created by applying AHC clustering to a sample of the documents of the collection. This sampling is risky when one is possibly interested in small clusters, as they may not be represented in the sample. Finally, we note that neither of these algorithms is incremental.

In contrast to STC, all the mentioned algorithms treat a document as a set of words and not as an ordered sequence of words, thus losing valuable information. Phrases have long been used to supplement word-based indexing in IR systems (*e.g.*, Buckley et. al. 95). The use of lexical atoms and of syntactic phrases has been shown to improve precision without hurting recall (Zhai et. al., 95). Phrases generated by simple statistical approaches (e.g., contiguous non-stopped words) have also been successfully used (Salton et. al, 75; Fagan, 87; Hull et. al., 97). Yet these methods have not been widely applied to document clustering. The only example known to the authors is the use of the co-appearance of pairs of words as the attributes of the documents' vector representations (Maarek and Wecker, 94).

On the Web, there are some attempts to handle the large number of documents returned by search engines. Many search engines provide query refinement features. AltaVista, for example, suggests words to be added or to be excluded from the query. These words are organized into groups, but these groups do not represent clusters of documents. The Northern Light search engine (*www.nlsearch.com*), provides "Custom Search Folders", in which the retrieved documents are organized. Each folder is labeled by a single word or a two-word phrase, and is comprised of all the documents containing the label. Northern Light does not reveal the method used to create these folders nor its cost.

3. SUFFIX TREE CLUSTERING

Suffix Tree Clustering (STC) is a linear time clustering algorithm that is based on identifying the phrases that are common to groups of documents. A phrase in our context is an ordered sequence of one or more words. We define a *base cluster* to be a set of documents that share a common phrase.

STC has three logical steps: (1) document "cleaning", (2) identifying base clusters using a suffix tree, and (3) combining these base clusters into clusters.

3.1. Step 1 - Document "Cleaning"

In this step, the string of text representing each document is transformed using a light stemming algorithm (deleting word prefixes and suffixes and reducing plural to singular). Sentence boundaries (identified via punctuation and HTML tags) are marked and non-word tokens (such as numbers, HTML tags and most punctuation) are stripped. The original document strings are kept, as well as pointers from the beginning of each word in the transformed string to its position in the original string. This enables us, once we identify key phrases in the transformed string, to display the original text for enhanced user readability.

3.2. Step 2 - Identifying Base Clusters

The identification of base clusters can be viewed as the creation of an inverted index of phrases for our document collection. This is done efficiently using a data structure called a *suffix tree* (Weiner, 73; Gusfield, 97). This structure can be constructed in time linear with the size of the collection, and can be constructed incrementally as the documents are being read (Ukkonen, 95). The idea of using a suffix tree for document clustering was first introduced in (Zamir et. al., 97). Here we present an improved clustering algorithm, which introduces the merger of base clusters (step three of the STC algorithm), and compare it using standard IR methodology to classical clustering methods in the Web domain.

A suffix tree of a string *S* is a *compact trie* containing all the suffixes of *S*. We treat documents as strings of words, not characters, thus suffixes contain one or more whole words. In more precise terms:

- A suffix tree is a rooted, directed tree.
- Each internal node has at least 2 children.
- Each edge is labeled with a non-empty sub-string of *S* (hence it is a *trie*). The label of a node in defined to be the concatenation of the edge-labels on the path from the root to that node.
- No two edges out of the same node can have edge-labels that begin with the same word (hence it is *compact*).
- For each suffix *s* of *S*, there exists a *suffix-node* whose label equals *s*.

The suffix tree of a collection of strings is a compact trie containing all the suffixes of all the strings in the collection. Each suffix-node is marked to designate from which string (or strings) it originated from (*i.e.*, the label of that suffix-node is a suffix of that string). In our application, we construct the suffix tree of all the sentences of all the documents in our collection.

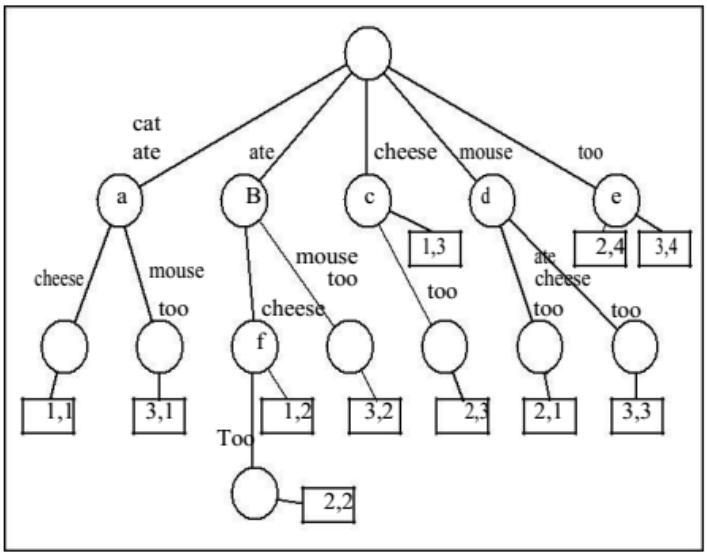

Fig2. *The suffix tree of the strings "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too"*

Figure 2 is an example of the suffix tree of a set of strings - "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too". The nodes of the suffix tree are drawn as circles. Each suffix-node has one or more boxes attached to it designating the string(s) it originated from. The first number in each box designates the string of origin (1-3 in our example, by the order the strings appear above); the second number designates which suffix of that string labels that suffix-node. Several of the nodes in the Figure are labeled *a* through *f* for further reference.

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; the set of documents tagging the suffix-nodes that are descendants of the node make up the document group. Therefore, each node represents a base cluster. Furthermore, all possible base clusters (containing 2 or more documents) appear as nodes in our suffix tree. Table 3 lists the six marked nodes from the example shown in Figure 2 and their corresponding base clusters.

Table3. *Six nodes from the example shown in Figure 2 and their corresponding base clusters.*

Node	Phrase	Documents
a	cat ate	1,3
b	ate	1,2,3
c	cheese	1,2
d	mouse	2,3
e	too	2,3
f	ate cheese	1,2

Each base cluster is assigned a score that is a function of the number of documents it contains, and the words that make up its phrase.

The score *s (B)* of base cluster *B* with phrase *P* is given by:

$$S(B)= |B|.f(|P|)$$

where $|B|$ is the number of documents in base cluster *B*, and $|P|$ is the number of words in *P* that have a non-zero score (*i.e.*, the effective length of the phrase). We maintain a stoplist that is supplemented with Internet specific words (e.g., “previous”, “java”, “frames” and “mail”). Words appearing in the stoplist, or that appear in too few (3 or less) or too many (more than 40% of the collection) documents receive a score of zero. The function *f* penalizes single word phrases, is linear for phrase that are two to six words long, and becomes constant for longer phrases.

3.3. Step 3 - Combining Base Clusters

Documents may share more than one phrase. As a result, the document sets of distinct base clusters may overlap and may even be identical. To avoid the proliferation of nearly identical clusters, the third step of the algorithm merges base clusters with a high overlap in their document sets (phrases are not considered in this step). In Figure 1, for example, the top cluster resulted from merging the two base clusters labeled "Puerto Rico" and "Latin Music" based on their document sets overlap.

We define a binary similarity measure between base clusters based on the overlap of their document sets. Given two base clusters B_m and B_n, with sizes $|B_m|$ and $|B_n|$ respectively, and $|B_m \cap B_n|$ representing the number of documents common to both base clusters, we define the similarity of Bm and Bn to be 1 iff:

- $|B_m \cap B_n|/|B_m|>0.5$
- $|B_m \cap B_n|/|B_n|>0.5$ and

Otherwise, their similarity is defined to be 0.

Next, we look at the *base cluster graph*, where nodes are base clusters, and two nodes are connected iff the two base clusters have a similarity of 1. A cluster is defined as being a connected component in the base cluster graph. Each cluster contains the union of the documents of all its base clusters. Figure 4 illustrates the base cluster graph of the six base clusters in Table 3. There is a single cluster in this example.

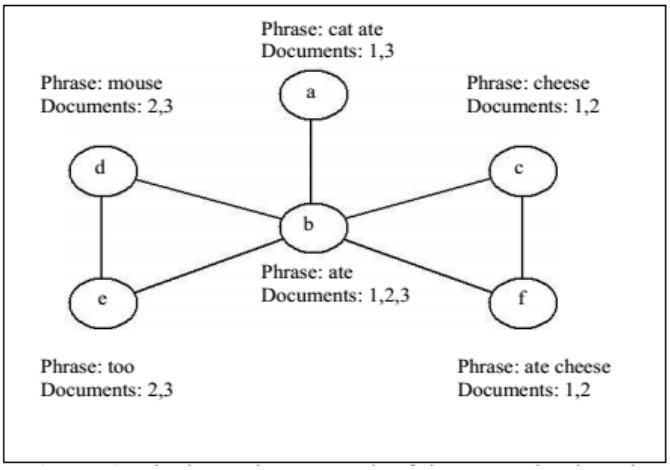

Fig4. *The base cluster graph of the example given in Figure 2 and in Table 3. In this example there is one connected component, therefore one cluster. Notice that if the word ate had been in our stoplist, the base cluster b would have been discarded as it would have had a score of 0, and then we would have had three connected components in the graph, representing three clusters.*

In essence, we are clustering the base clusters using the equivalent of a single-link clustering algorithm where a predetermined minimal similarity between base clusters serves as the halting criterion. This clustering algorithm is incremental and order independent. We do not encounter the undesired chaining effect of single-link clustering because we use it in the domain of base clusters where we typically find only small connected components.

The STC algorithm is incremental. As each document arrives from the Web, we "clean" it and add it to the suffix tree. Each node that is updated (or created) as a result of this is tagged. We then update the relevant base clusters and recalculate the similarity of these base clusters to the rest of the base clusters. Finally, we check if the changes in the base cluster graph result in any changes to the final clusters.

To keep the cost of this last step constant, we don't check the similarity of the modified base clusters with all other base clusters, but only with the *k* highest scoring base clusters (we take *k* to be 500 in our experiments). The cost of "cleaning" the documents is obviously linear with the collection size. The cost of inserting documents into the suffix tree is also linear with the collection size, as is the number of nodes that can be affected by these insertions. Thus the overall time complexity of STC is linear with regard to the collection size.

The final clusters are scored and sorted based on the scores of their base clusters and their overlap. As the final number of clusters can vary, we report only the top few clusters. Typically, only the top 10 clusters are of interest. For each cluster we report the number of documents it contains, and the phrases of its base clusters.

The goal of a clustering algorithm in our domain is to group each document with others sharing a common topic, but not necessarily to partition the collection. It has been claimed that it is artificial to force each document into only one cluster, as documents often have several topics (Hearst, 98). Such a constraint could decrease the usefulness of the clusters produced. Allowing a document to appear in more than one cluster acknowledges that documents are complex objects which may be grouped into multiple potentially overlapping, but internally coherent, groups. This is actually the reason many IR system use some form of dot-product document similarity measure (as opposed to Euclidean distance, for example): it allows a document to be similar to multiple distinct documents or centroids that could in turn be very dissimilar from each other.

In STC, as documents may share more than one phrase with other documents, each document might appear in a number of base clusters. Therefore a document can appear in more than one cluster. Note that the overlap between clusters cannot be too high, otherwise they would have been merged into a single cluster. In the example shown in Figure 1 (results of STC on the query "salsa"), a cluster relating to salsa recipes was produces as well as a cluster relating to companies selling salsa products. Several documents were correctly placed in both clusters as they included both recipes and information about the companies marketing them.

The STC algorithm does not require the user to specify the required number of clusters. It does, on the other hand, require the specification of the threshold used to determine the similarity between base clusters (0.5 in our example and experiment). However, we found that the performance of STC is not very sensitive to this threshold, unlike AHC algorithms that showed extreme sensitivity to the number of clusters required.

4. EXPERIMENTS

In order to evaluate STC we compared it both to the original ranked list of the search engine and to other clustering algorithms. The algorithms used in the comparison were group-average agglomerative hierarchical clustering (which will be referred to as GAHC), K-Means, Buckshot, Fractionation and Single-Pass. The GAHC algorithm was chosen as it is commonly used; the rest were chosen as they are fast enough to be contenders for on-line clustering.

For our experiments, we constructed document collections by saving the results of different queries to the MetaCrawler search engine. We chose not to use standard IR collections, as we were interested in the performance of document clustering *on the Web.* As the MetaCrawler is a meta search engine, (i.e. it routes queries to several other search engines and then collates their results), we assume its search results and the snippets it returns are representative of Web search engines.

4.1. Effectiveness for Information Retrieval

As we did not use a standard IR corpus, we were forced to generate our own queries and relevance judgments, though we are aware that this could lead to a bias in our results. To counteract any potential bias, we plan to publish our data set on the Web to allow independent validation and replication of our experiments.

The process was as follows: We first defined 10 queries by specifying their topics (*e.g.*, "black bear attacks") and their descriptions (*e.g.*, "we are interested in accounts of black bear attacks on humans or information about how to prevent such attacks"). The words appearing in each query's topic field were used as keywords for a Web search using the MetaCrawler search engine. We generated 10 collections of 200 snippets from the results of these queries. For each snippet returned by the search engine, we also downloaded its original document from the Web, thus generating 10 collections of 200 Web documents. We manually assigned a relevance judgment (relevant or not) to each document in these collections based on the queries' descriptions. On average there were about 40 relevant documents for each query.

In our first experiment we applied the various clustering algorithms to the document collections and compared their effectiveness for information retrieval. Specifically, we used the results of the clustering algorithms to reorder the list of documents returned by the search engine, according to a variation of the method laid out in (Hearst and Pedersen, 96; Schütze and Silverstein, 97), which assumes that the user is able to select the cluster with the highest relevant document density.[2]

As different clustering algorithms tend to produce clusters of different sizes and we did not want this to artificially influence the comparison between them, we considered only a constant number of documents (chosen by starting with the top cluster and working our way down through subsequent clusters until we reach 10% of the document collection); the remaining 90% of the documents were considered irrelevant. Thus, while (Hearst and Pedersen, 96) always pick a single cluster, we may pick more than one, or only a fraction of one in the case where the top cluster contains more than 10% of the documents in the collection. When this reordering method is applied to overlapping clusters, one might consider the same document more than once. Therefore, if a document is seen an additional time it is deemed irrelevant, as re-viewing it does not help the user.

While actual user behavior is quite complex and idiosyncratic, we believe that our methodology provides a better model of user behavior. A user picking a large cluster to investigate first might not scan all the documents in it, while a user picking a small cluster first might proceed to a second cluster once she's done with the first.

All algorithms (including STC) were run to produce the same number of clusters (10 in our experiments). This is necessary to allow a fair comparison of the different algorithms. The algorithms use the same parameter settings wherever relevant (*e.g.*, the minimal cluster size), and were optimized on a separate data set. Figure 5 compares the average precision of the various clustering algorithms with that of the original ranked list, averaged over the 10 Web document collections.

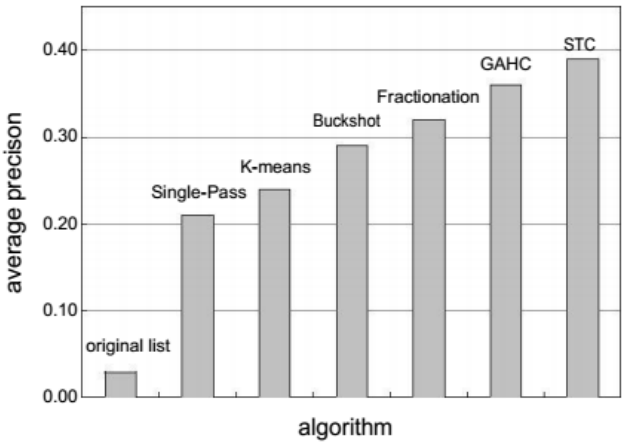

Fig5. *The average precision of the clustering algorithms and of the original ranked list returned by the search engine averaged over the 10 original document collections.*

As seen in Figure 5, the STC algorithm scored highest in this experiment. We believe that these positive results are due in part to STC's use of phrases to identify clusters and due to the fact that it naturally allows overlapping clusters. In our experiment each document was placed in 2.1 clusters on average and 72% of the documents were placed in more than one cluster. Regarding the use of phrases, 55% of the base clusters were based on phrases containing more than one word.

To measure the impact of these features on STC's performance, we ran an ablation study in which we created two hobbled variants of the STC algorithm. In the first variant - *STC-no-overlap* - we ran a post-processing phase that eliminated any overlap between the clusters by removing each document that was placed in several clusters from all but one cluster, the one whose centroid was the closest to the document. In the second variant - *STC-no-phrases* - we allowed STC to use only single word phrases. The performance of these variations appears in Figure 6.

We see that both cluster overlap and multi-word phrases are critical to STC's success. Phrases are key because they are the basis for identifying cohesive clusters; overlap is key because we have no means of deciding which phrase in a document ought to determine its assignment to a cluster. Overlap enables the document to potentially participate in all clusters that are based on shared phrases from that iteration of the algorithm. Again the impact on performance, document. As seen in Figure 8, is quite small.

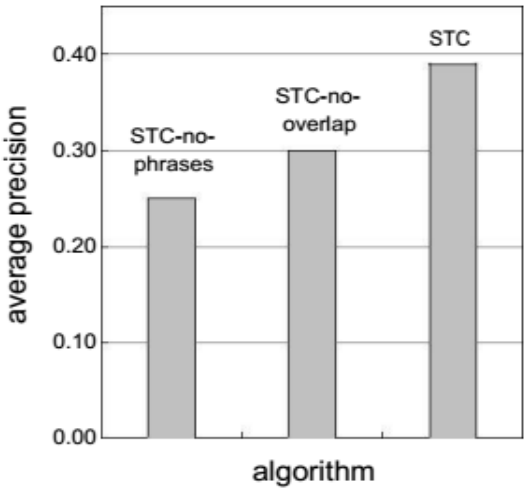

Fig6. *The average precision on the 10 document collections of the variants of STC: STC-no-overlap which forces the results into a true partition and STC-no-phrases which uses only single word phrases.*

Next we considered whether the use of phrases or overlap could be used to improve the average precision of standard clustering algorithms in a straightforward manner. For instance, one might argue that the suffix tree is merely performing a term extraction of sorts on the documents. Nevertheless, we measured the impact of introducing multi-word phrases as additional attributes on the performance of vector-based clustering algorithms. We examined two vector-based clustering algorithms - GAHC

and K-Means - and compared the standard algorithm which uses only single words as document vector attributes, to a modified version which uses phrases (single- and multi-word) identified by a suffix tree as attributes of the document vectors. This experiment was run on the original Web documents collections. The results in Figure 7 show that the modification can have either a positive or a negative impact on the performance of the vector-based algorithms, but this effect is not as dramatic as the impact of multi-word phrases on the STC algorithm. More experimentation is needed to understand this issue further.

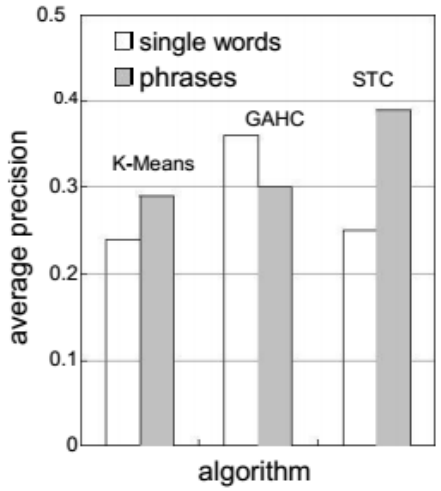

Fig7. *The average precision of clustering algorithms with and without the use of phrases identified by a suffix tree. Whereas STC's performance degraded substantially when multi-word phrases were disallowed, the modification did not have a substantial or consistent effect on the vector-based algorithms.*

Next, we examined the effect of allowing overlapping clusters on the different algorithms. We modified Buckshot and K-Means to allow overlapping clusters by allowing each document to be placed in more than one cluster in the last

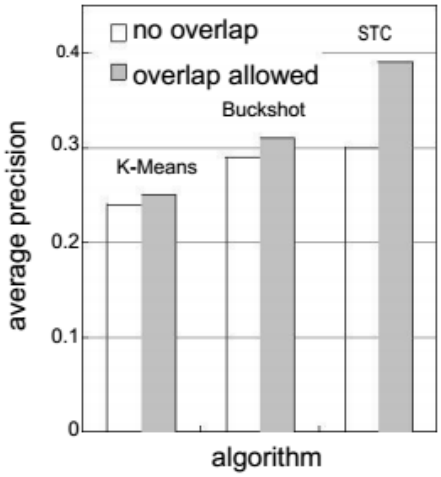

Fig8. *The average precision of overlap-producing clustering algorithms compared to their non-overlapping versions. The Buckshot and K-Means algorithms were chosen as they can easily be adapted to produce overlapping clusters.*

It is interesting to note that the degree of cluster overlap produced by the algorithms is quite different. Table 9 presents the average number of clusters that a document is placed in by the three overlap-producing algorithms. We calculated this statistic separately for relevant and for irrelevant documents. As can be seen in Table 9, the degree of overlap that STC produces is much greater than that of the two other algorithms. Given an average precision metric, allowing a document to appear in multiple clusters is only advantageous if that document is relevant, because it tends to increase the density of relevant documents that the user sees (on average). On the other hand, placing an irrelevant document in multiple clusters can hurt cluster quality. Therefore, the ratio of the two statistics in rows one and two of Table 9 could be viewed as an indication of the benefit of overlapping clusters. Indeed, the average precision results in Figure 8 correlate with this ratio - the algorithm most effected by allowing overlapping clusters in Figure 8 (STC) has the highest ratio, etc.

Table9. *The average number of clusters each document is placed in by the three overlap-producing algorithms. We calculated this statistic separately for relevant and for irrelevant documents. The ratio of these two statistics could be viewed as an indication of the benefits of overlapping clusters.*

	K-Means	Buckshot	STC
Avg. num of clusters:	1.40	1.40	2.60
Relevant document.			
Avg. num of clusters:	1.55	1.35	1.90
Irrelevant document			
Ratio of the above	0.90	1.04	1.37

It's clear that while multi-word phrases and overlap are critical to the success of STC (Figure 6), these elements of STC cannot be plugged in willy-nilly into clustering algorithms (Figures 7 and 8); they are an inextricable part of the novel STC algorithm.

We also conjecture that the multi-word phrases of the base clusters are very useful in conveying the clusters' contents to the user. For each cluster we display the phrases of the base clusters it contains (in our experiments, each cluster contained an average of five base clusters), as well as additional words that appear most frequently in the cluster (these words are identified only after the cluster has been formed). Again, user studies will have to be carried out to corroborate this conjecture.

4.2. Snippets versus Whole Document

A major issue in the feasability of clustering Web search engine results is whether similar performance could be produced when clustering only the snippets returned by the search engines. Figure 10 shows how clustering snippets affects the performance of the different clustering algorithms. We continue to rely on the relevance judgements from the previous experiments.

As shown in Figure 10, the decrease in the quality of the clusters is apparent but relatively small. This is surprising as, in our experiments, a Web document contained 760 words on average (220 words after eliminating stoplist words or words appearing in too few or too many documents), while a snippet contained 50 words on average (20 words after word elimination). One explanation is that the snippets represent attempts by the search engines to extract meaningful phrases from the original documents. Therefore the snippets contain phrases that help in the correct clustering of the document, and do not contain some of the "noise" present in the original documents that might cause misclassification of the documents. These results intimate earlier findings by Schütze and Silverstein, which showed that cluster quality is not adversely affected by truncating the vector representation of documents in standard IR collections (Schütze and Silverstein, 97).

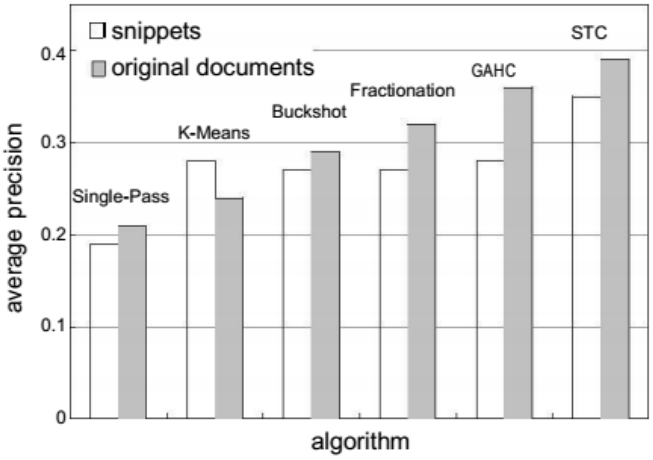

Fig10. *The average precision of clustering algorithms on the snippet collections compared with the average precision on the original Web documents collections.*

4.3. Execution Time

We measured the execution time of the various clustering algorithms while clustering snippet collections of various sizes (100 to 1000 snippets). The results are shown in Figure 11. Each reported time is averaged over 10 collections of the same size. The times were measured using a Linux machine running on a Pentium 200 processor.

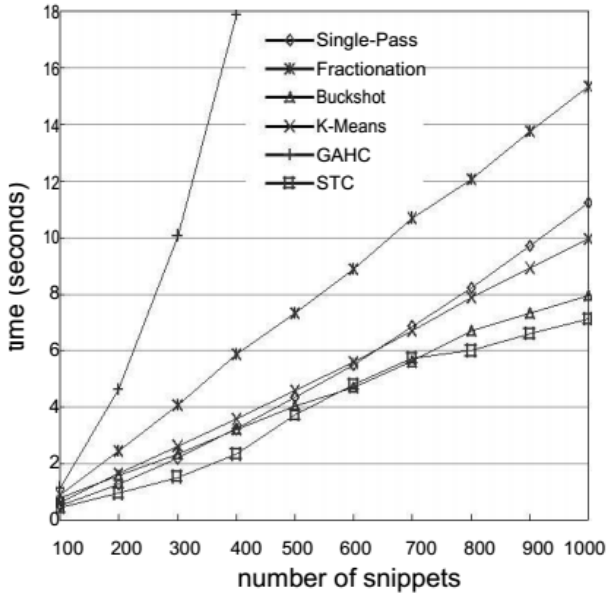

Fig11. *Execution time (in seconds) of the different clustering algorithms on snippet collections as a function of the collection size. In practice, STC is even faster than this graph suggests due to its incremental nature. STC can carry out the clustering process as documents are arriving from the Web. In our experiments, Meta Crawler-STC returns its results to the user a mere 0.01 seconds after the last document is received by Meta Crawler!*

It is plain that only near linear time algorithms can produce clusters for collections of hundreds of documents fast enough for true on-line interaction. STC is shown to be just as fast, if not faster, than other linear time algorithms. However, this comparison is conservative as is does not take into account the incremental nature of STC. As argued in the introduction, we believe document clustering should be done on a machine separate from the search engine server, which will receive search engine results over the Web and output clusters to the user. Because STC is incremental, it can use the "free" CPU time in which the system is waiting for the search engine results to arrive over the Web. Therefore, if the Web delay is on the order of 10 seconds, STC would produce results instantaneously after the last document arrives, while the non-incremental algorithms will only start their computations. Being incremental also enables the system to instantaneously display results when an impatient user interrupts the clustering algorithm, and allows it to be used for event detection and tracking tasks.

5. CONCLUSION

The main contributions of this paper are (1) the identification of the unique requirements of document clustering of Web search engine results, (2) the definition of STC - an incremental, *O(n)* time clustering algorithm that satisfies these requirements, and (3) the first experimental evaluation of clustering algorithms on Web search engine results, forming a baseline for future work.

Overall, our preliminary experiments are encouraging and suggest that fast document clustering algorithms (such as STC) can indeed be useful in clustering search engine results. Moreover, it appears that clustering the snippets returned by search engines is a reasonable and speedy alternative to downloading the original documents. Needless to say, a user study is needed to demonstrate the direct usefulness of clustering search engine results to support information access tasks on the Web. Further experiments are also necessary to confirm STC's apparent advantage over existing clustering algorithms.

To gather data from "live users", we have fielded the MetaCrawler-STC system on the Web. We have instrumented the system to log the queries made, the clusters found, and the links followed by users. In future work we intend to report statistics on the behavior of the system in practice, and to perform a controlled user study to further contrast STC with the ranked-list presentation and with other clustering methods.

REFERENCES

R. B. Allen, P. Obry and M. Littman. An interface for navigating clustered document sets returned by queries. In *Proceedings of the ACM Conference on Organizational Computing Systems*, pages 166-71, 1993.

C. Buckley, G. Salton, J. Allen and A. Singhal. Automatic query expansion using SMART: TREC-3. In: D. K. Harman (ed.), *the Third Text Retrieval Conference (TREC-3)*. U.S. Department of Commerce, 1995.

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

D. R. Cutting, D. R. Karger and J. O. Pedersen. Constant interaction-time Scatter/Gather browsing of large document collections. In *Proceedings of the 16th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 126-35, 1993.

W. B. Croft. *Organizing and searching large files of documents.* Ph.D. Thesis. University of Cambridge, October 1978.

J. L. Fagan. *Experiments in automatic phrase indexing for document retrieval: a comparison of syntactic and non-syntactic methods*. Ph.D. Thesis, Cornell University, 1987.

D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*, chapter 6. Cambridge University Press, 1997.

M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: Scatter/Gather on retrieval results. In *Proceedings of the 19th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 76-84, 1996.

M. A. Hearst. The use of categories and clusters in information access interfaces. In T. Strzalkowski (ed.), *Natural Language Information Retrieval*, Kluwer Academic Publishers, to appear.

D. R. Hill. A vector clustering technique. In Samuelson (ed.), *Mechanised Information Storage, Retrieval and Dissemination*, North-Holland, Amsterdam, 1968.

D. A. Hull, G. Grefenstette, B. M. Schulze, E. Gaussier, H. Schütze and L. O. Pedersen. Xerox TREC-5 site report: routing, filtering, NLP, and Spanish tracks. In: D. K. Harman (ed.), *The Fifth Text Retrieval Conference (TREC-5)*. NIST Special Publication, 1997.

A. V. Leouski and W. B. Croft. An evaluation of techniques for clustering search results. Technical Report IR-76, Department of Computer Science, University of Massachusetts, Amherst, 1996.

Y. S. Maarek and A. J. Wecker. The Librarian's Assistant: automatically organizing on-line books into dynamic bookshelves. In *Proceedings of RIAO'94*, 1994.

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.

E. Rasmussen. Clustering Algorithms. In W. B. Frakes and R. Baeza-Yates (eds.), *Information Retrieval*, pages 419-42. Prentice Hall, Eaglewood Cliffs, N. J., 1992.

J. J. Rocchio, *Document retrieval systems - optimization and evaluation*. Ph.D. Thesis, Harvard University, 1966.

G.Salton, C. S. Yang and C. T. Yu. A theory of term importance in automatic text analysis. JASIS, 26(1):33-44, 1975.

H.Schütze and C. Silverstein. Projections for efficient document clustering. In Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 74-81, 1997.

E. Selberg and O. Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 4th World Wide Web Conference*, 1995.

J. Shakes, M. Langheinrich and O. Etzioni. Ahoy! the home page finder. In *Proceedings of the 6th World Wide Web Conference*, 1997.

C. Silverstein and J. O. Pedersen. Almost-constant time clustering of arbitrary corpus subsets. In *Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 60-66, 1997. *Proceedings of the TDT Workshop*, University of Maryland, College Park, MD, October 1997.

E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249-60, 1995.

C. J. van Rijsbergen, Information Retrieval, Butterworths, London, 2nd ed., 1979.

E. M. Voorhees. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing and Management*, 22:465-76, 1986.

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

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

AUTHORS' BIOGRAPHY

Ganesh Kr. Yadav

Born in 14th July 1984 at Allahabad (UP). He had done B.Tech (Computer Science & Engg.) from Babu Banarsi das Institute of Technology, Ghaziabad, M. Tech (IT)* SITM, Lucknow,. Work Experience: He had been in different institution / university like Meerut Institute of Engineering College, Meerut, (U.P), Meerut International Institute of Technoly, Meerut, SR Group of Institutions, Jhansi (U.P.). He has published 6 papers in national Conference. Area of Interest in Computer Graphics, Data/Web Mining, Software Engg, Mobile Computing.

Amit Kumar

Born in 2 january 1983 in patna(Bihar) . He had done MCA (Master of Computer Applications.) from Jaipuria Institute of management. Work Experience: He had been in different institution like CDAC Hyderabd, (Combined Andhra Pradesh), S Group of Institutions, Jhansi (U.P.)We Have Attended Three National Conferences. Area of Interest in Computer Graphics, Data/Web Mining, Software Engg, Mobile Computing.